Verifying Table-Based Elections

David Basin, Jannik Dreier, Sofia Giampietro, Saša Radomirović

Verifying Table-Based Elections

David Basin
Dept. of Computer Science, ETH Zürich
Zürich, Switzerland
basin@inf.ethz.ch

Jannik Dreier
Université de Lorraine, Inria, CNRS, Loria, UMR 7503
Vandoeuvre-lès-Nancy, France
jannik.dreier@loria.fr

Sofia Giampietro
Dept. of Computer Science, ETH Zürich
Zürich, Switzerland
sofia.giampietro@inf.ethz.ch

Saša Radomirović
Dept. of Computer Science, Heriot-Watt University
Edinburgh, UK
sasa.radomirovic@hw.ac.uk

ABSTRACT

Verifiability is a key requirement for electronic voting. However, the use of cryptographic techniques to achieve it usually requires specialist knowledge to understand; hence voters cannot easily assess the validity of such arguments themselves. To address this, solutions have been proposed using simple tables and checks, which require only simple verification steps with almost no cryptography.

This simplicity comes at a cost: numerous verification checks must be made on the tables to ensure their correctness, raising the question whether the success of all the small verification steps entails the overall goal of end-to-end verifiability while preserving vote secrecy. Do the final results reflect the voters' will? Moreover, do the verification steps leak information about the voters' choices?

In this paper, we provide mathematical foundations and an associated methodology for defining and proving verifiability and voter privacy for table-based election protocols. We apply them to three case studies: the Eperio protocol, Scantegrity, and Chaum's Random-Sample Election protocol. Our methodology helps us, in all three cases, identify previously unknown problems that allow an election authority to cheat and modify the election outcome. Furthermore, it helps us formulate and verify the corrected versions.

CCS CONCEPTS

• **Security and privacy → Logic and verification**; **Privacy protections**; **Privacy-preserving protocols**.

KEYWORDS

Protocol verification, Elections, Verifiability

ACM Reference Format:
David Basin, Jannik Dreier, Sofia Giampietro, and Saša Radomirović. 2021. Verifying Table-Based Elections. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3460120.3484555

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484555

1 INTRODUCTION

Problem context. Secure electronic voting (e-voting) has been a topic of active research since the 1980s [5, 10, 16] with clear practical relevance. As security is a paramount concern, governments have imposed requirements on the verifiability of the election results. For example, in Switzerland, when more than 30% of the electorate votes electronically, then voters must be able to check that their vote was recorded as cast ("individual verifiability"). Moreover, when more than 50% vote electronically, then auditors must be able to determine that the tally has not been manipulated ("universal verifiability") [8]. The questions arise of how such checks can best be supported, how to verify that the checking procedures used actually work as intended and themselves cannot be manipulated, and whether information about voter choices is leaked.

Our focus is on voting protocols with verification procedures based on *data tables* accompanied by simple *checks* that enable the voters and auditors to verify the election results. For example, a voter may be asked to check that a table entry matches her own vote selection, or that some values sum up correctly. These checks are simple for voters to understand and carry out. This is in contrast to more heavy-weight cryptographic protocols, which use mechanisms like homomorphic encryption [24] or mix networks employing zero-knowledge proofs [3, 30]. With such procedures, the computations made are relatively complex and evidence must be checked by programs rather than humans. The importance of using simple, comprehensible procedures for increasing transparency and the electorate's trust has also been emphasized by others [4, 7] and is a legal requirement in some countries. For example, the German Federal Constitutional Court ruled that political elections must be verifiable by everyone and without specialist knowledge [18].

But even using tables with simple operations and checks does not guarantee that these checks taken together are actually sufficient to ensure that the election result is correct, and proving the latter can be complex. Moreover, elections typically have conflicting requirements concerning the results' integrity and the voters' privacy. Correctness necessitates resolving this tension and determining that all the checks, taken together, satisfy all the given requirements. Prior to this work, a general proof methodology for this, in the context of table-based election protocols, was lacking.

Approach taken. We present mathematical foundations and an associated methodology for establishing that verification checks for table-based protocols ensure the correctness and privacy of the election's results. Our foundations are based on tables, which are

multisets of tuples, and a small collection of operators for manipulating, querying, and combining them. Protocols may be built from tables using simple cryptographic operations such as commitments and cut & choose steps. Our foundations are particularly well-suited for cut & choose protocols, where different permuted copies of a table are used, and each copy reveals just part of its data. We use our operators to define general notions relevant for such elections, for example when these partially revealed tables can be combined in a consistent way or when two tables are indistinguishable, given a set of their subtables or other properties.

Given a voting protocol, its analysis proceeds as follows:

(1) Specify a global *integrity property* ϕ of the election, for example that the final tally reflects the voters' choices.
(2) Provide a set $\Psi = \{\psi_1, \ldots, \psi_n\}$ of properties of the tables used in the election, which correspond to the simple checks agents must make as described by the protocol, and prove that together, they are sufficient to establish ϕ. Hence, if the check of each $\psi_i \in \Psi$ succeeds, then the electorate can be assured of the correctness of the overall election results (**soundness**), even in presence of malicious participants.
(3) Verify that if the global integrity property ϕ holds, then the checks will succeed (**completeness**).
(4) Prove that the checks do not violate the voters' privacy.

These steps are general and can be applied to any table-based protocol to ensure both integrity and privacy. Moreover, our table operators directly support formulating and proving the properties in these steps.

To illustrate this, we present three case studies. The first case study, which is also our running example, analyzes Eperio [22]. This protocol can be seen as a variant of e-voting protocols like Punchscan [31], Scantegrity [12], and Aperio [21] and is based on simple checks on tables where each check is made on a randomly chosen subset of table elements. We have chosen Eperio to illustrate our methodology because it is formulated directly on tables. In our second case study, we obtain similar modeling results for the better-known protocol Scantegrity [12].

The previous analysis of Eperio [22] focused on the probability that each individual step ψ_i detects a malicious execution. However, such analysis does not address the correctness of the entire procedure when putting all the checks together. Our methodology instead analyzes table-based protocols in a possibilistic setting, and aims to show the correctness of the verification steps taken together. Our analysis reveals that even if all the checks in Eperio succeed, some initial assumptions or additional checks are needed, as otherwise there are weaknesses that can be exploited by a malicious election authority. Furthermore, our methodology reveals that Scantegrity is subject to the same weaknesses.

Our third case study is a table-based version of random-sample elections proposed by Chaum [9]. Chaum presents his protocol informally without proofs of his security claims. We apply our methodology to provide a rigorous analysis of his protocol. We again uncover critical weaknesses that would allow a malicious election authority to modify the election results and we provide additional checks to fix them with associated proofs.

Contributions. We provide both foundations and an associated methodology for reasoning about table-based elections. Prior to this work such a methodology was missing. Our foundations generalize operators from database theory to handle multisets and undefinedness and provide an expressive and concise formalism to specify and reason about table properties. Our methodology provides a sound way to decompose global correctness properties into locally checkable ones. We provide three substantial case studies that illustrate its scope. These examples also show that, despite the simplicity of the checks, table-based analysis is subtle. Our work shows how to make this analysis precise, thereby supporting correctness proofs or uncovering errors in proposed protocols.

One particularly novel aspect of our work, which should be of independent interest, is the design of table-based methods for analyzing and quantifying the privacy of elements in a table. We define a novel, combinatorial notion of privacy for the table-based setting, called G-indistinguishability, that measures an observer's ability to correlate elements in one column of a table with elements in another based on a fixed set of views and queries. This is well suited for table-based voting protocols where privacy may be breached by re-linking information from *within* a given data set and can also be used in other applications storing private data in tables. We use G-indistinguishability to provide sufficient conditions for a protocol to achieve privacy, and we compare this notion to symbolic equivalence-based definitions of privacy.

Organization. In Section 2 we present foundations for manipulating tables. In Section 3 we introduce our first case study, Eperio, and present protocol modeling. In Sections 4 and 5 we show how to verify integrity and privacy properties using Eperio as a running example. In Section 6 we obtain similar modeling results for the Scantegrity protocol. In Section 7 we sketch the application of our methodology to random sample elections, providing the full details in Section E of the appendix. We discuss related work in Section 8 and draw conclusions in Section 9.

All proofs of the lemmas and theorems stated in the paper are given in the Appendix.

2 TABLE OPERATIONS

2.1 Preliminaries

We start by defining basic operations on tables, generalizing standard notions from database theory to account for tables that are multisets and may contain missing information. We take a named perspective on tables, following [1]. Namely, a table T is a finite *multiset* of tuples (the table's rows), where the table's columns are uniquely named. We write $\mathrm{set}(T)$ for the set of tuples that occur at least once in T, $\#_T(t)$ to denote the multiplicity of a tuple t in T, and $|T|$ for the number of tuples in T, counting multiplicity. A table's rows and columns are unordered. The set of T's *column names* is called its *sort* and is denoted by $\mathrm{sort}(T)$. Given an underlying data domain *dom*, each column name may be associated with a subset of $dom \cup \{\bot\}$, also written as $dom_\bot$, e.g., representing (possibly unknown) strings or integers. We assume that *dom* does not contain the symbol $\bot$, which represents missing information. We refer to elements in *dom* as *known values* and to $\bot$ as an *unknown value*. Similarly, tuples and tables without $\bot$ are called *known* tuples and tables, respectively.

We view tuples as total functions from their sort to $dom_\bot$ where for a tuple t of sort N, $t(A)$ is t's value on the column name $A \in N$. We write tuples with angled brackets and include both the column

name and corresponding value, e.g., $\langle A\colon 3, B\colon 5\rangle$, which we consider to be the same tuple as $\langle B\colon 5, A\colon 3\rangle$. We say that a tuple s is a supertuple of a tuple t if t is the restriction of s to $\mathrm{sort}(t)$, i.e., $s|_{\mathrm{sort}(t)} = t$.

We denote by $\mathcal{T}$ the set of tables over a set dom and by $\mathcal{T}_\perp$ the set of tables over $dom_\perp$. For $a, b \in dom_\perp$, we write $a \doteq b$ if $a = b$, $a = \perp$, or $b = \perp$. We extend $\doteq$ to tuples and tables in the natural way. We denote multiset containment by $\subseteq^\#$ and by $\subseteq^\#_\perp$ the canonical extension to $dom_\perp$. Formally, for $S, T \in \mathcal{T}_\perp$, we write $S \subseteq^\#_\perp T$ if there is a total function $w\colon \mathrm{set}(S) \times \mathrm{set}(T) \to \mathbb{N}$ such that

(1) $\forall (s,t) \in S \times T\colon w(s,t) \neq 0 \Rightarrow s \doteq t$,
(2) $\forall s \in S\colon \sum_{t \in \mathrm{set}(T)} w(s,t) = \#_S(s)$, and
(3) $\forall t \in T\colon \sum_{s \in \mathrm{set}(S)} w(s,t) \leq \#_T(t)$.

This definition intuitively corresponds to a "total mapping" from the multiset S to the multiset T. However instead of defining the notion of a mapping on multisets, we defined multiset containment directly.

2.2 Table Queries and Properties

We write table queries in the SP-algebra [1], with operators for Selection and Projection, following a standard syntax. For example, the selection query $\sigma_{A=1}(T)$ returns the table that is the multiset of tuples in T that have a 1 in the column named A and the projection query $\pi_N(T)$ returns the projection of T to the columns with names in the set N. If $N \not\subseteq \mathrm{sort}(T)$, then $\pi_N(T)$ is undefined. Note that for selection, $\perp$ is treated as a literal value: $\sigma_{A=\perp}(T)$ is the multiset of tuples that have $\perp$ in the column A of T. To treat $\perp$ as a wildcard, we apply the selection operator with $\doteq$, e.g., $\sigma_{A \doteq 1} T$ is the multiset of tuples in T that have a 1 or a $\perp$ in column A of T. We omit parentheses when applying and composing operators and write $\alpha\beta T$ for $(\alpha \circ \beta)(T)$.

A table property is a set of tables. A table $T \in \mathcal{T}_\perp$ has a property $p \subseteq \mathcal{T}_\perp$ if $T \in p$. Notationally, we will often identify properties with table predicates and write $p(T)$. We also consider properties of *sets* of tables, e.g. $p(\mathcal{S})$, for a set of tables $\mathcal{S} \subseteq \mathcal{T}_\perp$. More generally, we will consider properties that are n-ary predicates, i.e. sets of n-tuples of tables or of sets of tables, and write $p(T_1, .., T_n)$ or $p(\mathcal{S}_1, .., \mathcal{S}_n)$.

For verifiability in voting, users and auditors often perform checks on individual tables or collections of tables. To model this we will work with properties whose membership is easily determined by simple checks, and we hence often identify checks (conducted either on tables or on sets of tables) and properties (of tables or of sets of tables respectively).

For a table $S \in \mathcal{T}_\perp$, the table property $shape_S := \{T \in \mathcal{T}_\perp \mid \mathrm{sort}(T) = \mathrm{sort}(S) \wedge |T| = |S|\}$ is the set of tables that have the same column names and number of tuples as S. A central property when combining potentially conflicting information from different sources is the set of all *known* tables that are horizontal (or column) or vertical (or row) extensions of a table $T \in \mathcal{T}_\perp$, or both. We denote this by $[T]$, where

$$[T] := \{T' \in \mathcal{T} \mid T \subseteq^\#_\perp \pi_{\mathrm{sort}(T)}(T')\}.$$

When information in a table T is associated with unique values, such as indices or keys of a sort N, we may then impose the additional side-condition that any extension of T preserves the uniqueness of tuples of sort N. We will write $[T]^N$ for those extensions of T where subtuples of sort N have multiplicity 1:

$$[T]^N := [T] \cap \{T' \in \mathcal{T} \mid N \subseteq \mathrm{sort}(T') \wedge \pi_N(T') = \mathrm{set}(\pi_N(T'))\}.$$

Example 1. Consider the following tables, where column names are written above the horizontal line:

$$T = \begin{array}{|cc|}\hline \textit{Ballot} & \textit{Candidate} \\ \hline 123 & \text{Asterix} \\ 337 & \text{Obelix} \\ \hline \end{array} \qquad S = \begin{array}{|ccc|}\hline \textit{Voter} & \textit{Ballot} & \textit{Candidate} \\ \hline a & 123 & \text{Asterix} \\ b & 337 & \text{Obelix} \\ c & 337 & \text{Dogmatix} \\ \hline \end{array}.$$

Then $S \in [T]$, but $S \notin [T]^{\{Ballot\}}$, because the ballot 337 occurs twice in S. For the same reason, $[S]^{\{Ballot\}} = \emptyset$. Note that $[T]^{\{Voter\}}$ is the set of all horizontal and vertical extensions of T with unique entries in the column *Voter*. This set contains S, but does not contain T itself because T does not have a *Voter* column. △

2.3 Table Indistinguishability

A central and novel notion in our work is a table's indistinguishability set with respect to a set of queries Q and a set of properties B. Intuitively, an indistinguishability set contains all possible explanations for what a table T could be, based on the observations an agent has obtained about T from the queries in Q and the properties in B that the agent knows T must have. An indistinguishability set is the rough analog for tables of an *anonymity set* from the privacy literature, where a subject is not identifiable within a group of subjects. We will use this notion to define privacy in Section 5.

Definition 1. *Let $T \in \mathcal{T}_\perp$ be a table over $dom_\perp$ and Q a set of queries and B a set of properties. The* indistinguishability set of T associated with Q and satisfying B *is the set of tables over dom where*

$$Ind_T(Q, B) \quad := \quad \{\, T' \in \mathcal{T} \mid \forall q \in Q\colon q(T') \doteq q(T) \;\wedge\; \forall b \in B\colon b(T') \,\}.$$

We write $Ind^{qu}_T(Q)$ for $Ind_T(Q, \emptyset)$.

Using Example 1, suppose only the columns *Ballot* and *Candidate* of the table S are queried. Then $Ind_S(\{\pi_{Ballot,Candidate}\}, \{shape_S\})$ contains all possibilities for voters to cast the three votes since this set consists of all tables of equal shape and equal subtuples of sort $\{Ballot, Candidate\}$ as S. The set $Ind^{qu}_S(\{\pi_{\mathrm{sort}(S)}\})$ is the set of known tables that horizontally extend S, allowing for arbitrary additional information to be associated with the three tuples in S.

3 TABLE PROTOCOLS

We use our foundations to model voting protocols where:

(1) voters' election data is stored in a table $\mathcal{E}$, containing for example the list of voters, candidates, and their choices;
(2) to enable verifiability, the election authority publishes parts of the election data (copies of $\mathcal{E}$ where only a subset of its rows or columns are revealed), denoted by tables $\mathcal{P}_{auth} \subseteq \mathcal{T}_\perp$, in a way that should preserve the voters' privacy; and
(3) voters and auditors verify that an integrity property ϕ (stating that votes are counted as intended) is satisfied by making a set of checks, denoted by $\Psi = \{\psi_1, ..., \psi_k\}$.

We will illustrate this on a running example, defining an additional table operator for combining tables in the process.

There is a large family of e-voting protocols adhering to the above scheme whose verification procedures are based on randomized checks and do not depend on cryptographic primitives. To preserve privacy, these checks are not made directly on the table $\mathcal{E}$ described in (1), but rather, as described in (2), on permuted copies of the table. Examples of such protocols are Scantegrity [12], Scantegrity II [11], Punchscan [31], and Eperio [22]. The protocols' verification procedures rely on cut & choose techniques. Namely, in Step (2), the authority does not know beforehand which data should be revealed and the correctness of the verification is based on the unpredictability of this choice. The protocols then only differ in how the permutations, receipts, and verification checks described in (3) are conducted. For example, Scantegrity uses a circuit-switchboard, Scantegrity II uses invisible ink on their ballots, while Punchscan uses two-layer ballots with holes in the top sheet. In contrast, Eperio performs the verification checks directly on the published tables.

For the sake of clarity, we have chosen Eperio as our running example to illustrate our methodology. Since Eperio's verification procedures are directly defined in terms of table checks, we can therefore focus on our methodology itself. We use Eperio to motivate and explain our formalism to represent and reason about tables. It also shows the benefit of such rigor as our analysis reveals weaknesses in Eperio's verification procedure, and provides clear and simple proofs of the soundness and privacy of a corrected version.

Afterwards, in Section 6, we apply our methodology to analyze Scantegrity. Once we abstract Scantegrity's verification mechanism based on circuit-switchboards, reducing it to table checks, we obtain similar results. We also analyze Chaum's Random Sample Election Protocol [9], which operates directly on tables. This protocol is interesting as it offers additional complexity by implementing a random choice among possible voters and using decoy ballots to prevent vote-buying. Prior to this work, a methodology for analyzing such table-based protocols was lacking.

3.1 Eperio's Election Procedure

U	Mark	Cand. (S)
Gs.1	●	Asterix
Gs.2	○	Obelix
	#Gs#	fill box to audit

U	Mark	Cand. (S)
Lf.1	○	Obelix
Lf.2	○	Asterix
	#Lf#	fill box to audit

Figure 1: Marked ballots with randomly ordered candidates.

Ballots in Eperio consist of two separable parts: the left half contains markable bubbles with unique serial numbers and the right half lists the corresponding candidates in a randomly permuted order as shown in Figure 1. To vote, voters mark their choice on the ballot. Once the marked ballot is scanned, the candidate list is removed and destroyed. The voter keeps the remaining part as a receipt. Alternatively, voters may abstain from voting and keep the entire ballot to audit the election. The ballot is still scanned, but marked as a so-called *print-audit* ballot. All scanned ballots are recorded by an authority in a table $\mathcal{E}$.

Observe that the vote associated to a ballot is determined only if *all* three columns are known. The Eperio verification procedure reveals two of the three columns, allowing voters to verify that their votes were recorded correctly, while not compromising vote secrecy. Using different row-shuffled copies of the table $\mathcal{E}$, the authority alternatively reveals the first two or the last two columns. Observe that by the uniqueness of the ballot numbers, the authority may not reveal the first two columns in one copy and the first and third column in another, without revealing the ballots' votes.

In more detail, the protocol proceeds as follows. The scanned ballots are recorded by the election authority in a table called the *Eperio table*, denoted $\mathcal{E}$, which is a $u \times 3$ table, where u is the product of the number of ballots and the number s of candidates in an election. The three columns are named U, M, and S. Column U contains the serial number of a ballot and a position in the range 1 through s. The third column S contains the candidate names. These two columns are fixed before the election. Column M is initially empty, but once ballots are recorded, it contains elements from the set $\{1, 0, -1\}$, formalizing whether the candidate (listed in the third column) was voted for (1), not voted for (0), or the ballot is selected for print auditing (-1). For example, the cast ballots from Figure 1 would result in the following Eperio table:

$$\mathcal{E} = \begin{array}{|ccc|} \hline U & M & S \\ \hline \text{Gs.1} & 1 & \text{Asterix} \\ \text{Gs.2} & 0 & \text{Obelix} \\ \text{Lf.1} & -1 & \text{Obelix} \\ \text{Lf.2} & -1 & \text{Asterix} \\ \hline \end{array}.$$

The table $\mathcal{E}$ is known only to the authority and is never published. Elections have three phases:

Phase 1 *Prior to the election.*
Before the election, the column M is empty. The authority generates x row-permutations of the table $\mathcal{E}$, where x is a fixed parameter. It generates next separate commitments for the first and the third column of each of the x copies of $\mathcal{E}$, denoted by $\mathcal{E}^1, \ldots, \mathcal{E}^x$. These commitments are published before the election.

Phase 2 *After the election, but prior to the audit.*
The votes are recorded in the column M and the x permutations of this column are published. Print-audited ballots are identified with a -1 entry in M. The authority generates and publishes a *linkage list* L. This is a table that specifies, for each row-permuted copy of $\mathcal{E}$, the row numbers in which the print-audited ballots' serial number and position can be found: L_j, the j-th column of the table L, contains the row numbers of the print-audited ballots in the table $\mathcal{E}^j$; see Figures 2 and 3.

Phase 3 *Audit.*
The auditors perform a cut & choose game over the x commitments of the permutations of the table $\mathcal{E}$ published in Phase 1 and perform a set of checks, which we denote by Ψ. For each commitment, either the column containing a permutation of U or the column containing a permutation of S is published. The corresponding permutation of M was already published in Phase 2. The linkage list is used to verify print-audited ballots. If the revealed column is a permutation of U, the linkage list is used to verify the serial number and position pair. If it is a permutation of S, it is used to verify the candidate order on the ballot.

U	M	L_1	…	L_x
⋮	⋮	⋮		⋮
Gs.1	-1	5	…	73
Gs.2	-1	20	…	88
Lf.1	-1	19	…	18
Lf.2	-1	3	…	100
⋮	⋮	⋮		⋮

Figure 2: Table composed of columns U and M, and linkage list L showing data structure for two print-audited ballots.

row	$\mathcal{E}_U^1$	$\mathcal{E}_M^1$	$\mathcal{E}_S^1$	row	$\mathcal{E}_U^x$	$\mathcal{E}_M^x$	$\mathcal{E}_S^x$
3	Lf.2	-1	Obelix	18	Lf.1	-1	Asterix
5	Gs.1	-1	Asterix	73	Gs.1	-1	Asterix
19	Lf.1	-1	Asterix	88	Gs.2	-1	Obelix
20	Gs.2	-1	Obelix	100	Lf.2	-1	Obelix

Figure 3: Entries pointed to by the linkage list in the first and last permuted copies of the Eperio table.

When the commitments are made, the authority does not know which columns it must reveal. Hence the auditors are ensured (with high probability) that *both* columns U and S are correct.

Formally, we use our table notation and we model the data that the authority publishes by $u \times 3$ tables $P_1, \ldots, P_x$ whose entries are the data in the publicly revealed columns and $\perp$ otherwise. We set $\mathcal{P}_{auth} := \{P_1, \ldots, P_x\} \subseteq \mathcal{T}_\perp$. Some tables will have unknown values in the U columns, others in the S columns. To model the information revealed by Eperio's linkage list table L, we replace unknown values in tuples that contain $M = -1$ in the tables $\mathcal{P}_{auth}$ by the information learned from the linkage list table.

The tally is a function, denoted *tal*, that is computed from the election table $\mathcal{E}$. For Eperio, $tal := \pi_S \sigma_{M=1}$ selects the rows in the Eperio table that satisfy $M = 1$ followed by a projection to the S column containing the candidate names.

In practice, the checks done by the voters and auditors are not performed on the secret election table $\mathcal{E}$, but rather on data published by the authority, represented by the tables in $\mathcal{P}_{auth}$. Observe that when the voting authority is honest, then each table in $\mathcal{P}_{auth}$ is a submultiset of $\mathcal{E}$ (as the order of rows in a table is immaterial). A dishonest authority may, however, publish arbitrary data, where some or all tables in $\mathcal{P}_{auth}$ are unrelated to any underlying table.

3.2 An Operator for Combining Information

To combine possibly conflicting tables, we define a novel operator $\bigcirc\!\!\!\!\wedge^{Key}$ on tables that merges data where the tuples of a given sort *Key* are used as unique keys. In the case of conflicting information, the operator returns the empty set.

Let *Key* be a set of column names and $\mathcal{R} \subseteq \mathcal{T}_\perp$ be a finite set of tables of possibly different sorts. We want to ensure that the tables in $\mathcal{R}$ may contain redundant, but not conflicting information. Namely, if $T_1, T_2 \in \mathcal{R}$ have a tuple t of sort *Key* in common, they must also have the super-tuple of t of sort $\text{sort}(T_1) \cap \text{sort}(T_2)$ in common. Formally, we define this combination as the intersection of all extensions of tables in $\mathcal{R}$ where subtuples of sort *Key* are unique,

$$\bigcirc\!\!\!\!\wedge^{Key}\mathcal{R} := \bigcap_{T \in \mathcal{R}} [T]^{Key}.$$

Example 2. Write $T \mathbin{\bigcirc\!\!\!\!\wedge^{Key}} S$ for $\bigcirc\!\!\!\!\wedge^{Key}\{T, S\}$ for tables T and S. Consider the following tables, where V contains voter identifiers,

$$R := \left|\begin{array}{ccc} U & M & S \\ \hline H.1 & 1 & yes \\ N.1 & 1 & no \end{array}\right|,\ S := \left|\begin{array}{ccc} U & M & S \\ \hline H.1 & 0 & yes \\ N.1 & 1 & no \end{array}\right|,\ T := \left|\begin{array}{ccc} V & U & M \\ \hline a & N.1 & 1 \\ b & K.2 & 0 \end{array}\right|.$$

Then:

(1) $R \mathbin{\bigcirc\!\!\!\!\wedge^{\{U,M\}}} S = [R]^{\{U,M\}} \cap [S]^{\{U,M\}} = [Y]^{\{U,M\}}$ where

$$Y = \left|\begin{array}{ccc} U & M & S \\ \hline H.1 & 1 & yes \\ H.1 & 0 & yes \\ N.1 & 1 & no \end{array}\right|.$$

That is, $R \mathbin{\bigcirc\!\!\!\!\wedge^{\{U,M\}}} S$ contains all tables that extend Y (both horizontally and vertically) and have unique subtuples of sort $\{U, M\}$.

(2) $R \mathbin{\bigcirc\!\!\!\!\wedge^{\{U\}}} S = \emptyset$, that is, R and S contain conflicting information if U is the unique key, as is the case in Eperio. R and S have the ballot id $H.1$ in common, but R contains the subtuple $\langle U\colon H.1, M\colon 1\rangle$, whereas S contains the subtuple $\langle U\colon H.1, M\colon 0\rangle$. So there is no extension that combines the information present in R and S while preserving the uniqueness of the subtuples of sort $\{U\}$.

(3) $S \mathbin{\bigcirc\!\!\!\!\wedge^{\{U,M\}}} T = [Z]^{\{U,M\}}$, where

$$Z = \left|\begin{array}{cccc} V & U & M & S \\ \hline \perp & H.1 & 0 & yes \\ a & N.1 & 1 & no \\ b & K.2 & 0 & \perp \end{array}\right|. \qquad \triangle$$

The following lemma, proved constructively, states that the result of $\bigcirc\!\!\!\!\wedge^{Key}$ computed over a set of known tables, where *Key* is a subsort of each table, always equals $[Z]^{Key}$ for some table Z. This includes the possibility that the result is the empty set.

Lemma 1. Let $T_1, \ldots, T_k \in \mathcal{T}$, and $Key \neq \emptyset$ be a set of column names such that *Key* is a subsort of each of these tables. Then there exists a table $Z \in \mathcal{T}_\perp$ such that $\bigcirc\!\!\!\!\wedge^{Key}\{T_1, \ldots, T_k\} = [Z]^{Key}$.

Hence if $\bigcirc\!\!\!\!\wedge^{U} \mathcal{P}_{auth} \neq \emptyset$, the tables in $\mathcal{P}_{auth}$ are consistent with each other, and the tables in $\bigcirc\!\!\!\!\wedge^{U} \mathcal{P}_{auth}$ represent all possible "Eperio tables" from which these subtables come from. Observe that tables in $\bigcirc\!\!\!\!\wedge^{U} \mathcal{P}_{auth}$ could also extend the tables in $\mathcal{P}_{auth}$ vertically. However, we are often interested in the minimal such extension.

Definition 2. *If $\mathcal{S} \subseteq \mathcal{T}$ is a set of tables, we denote by* $\min \mathcal{S}$ *the set of all minimal elements of $\mathcal{S}$, i.e.,* $\min \mathcal{S} \subseteq \mathcal{T}$ *is the largest subset of $\mathcal{S}$ with the property* $\forall T \in \mathcal{S}, T' \in \min \mathcal{S}\colon T' \in [T] \Rightarrow T = T'$.

For example, let $\{T_1, ..., T_k\} \subseteq \mathcal{T}$ and the sort *Key* be as in Lemma 1 so that $\mathbin{\bigcirc\!\!\!\!\wedge^{Key}}\{T_1, ..., T_k\} = [Z]^{Key}$ for some $Z \in \mathcal{T}_\perp$. Then $\min \mathbin{\bigcirc\!\!\!\!\wedge^{Key}}\{T_1, ..., T_k\}$ is the set of all tables obtained by replacing the unknown values of Z with known ones from the domain.

3.3 Modeling Table-Based Protocols

In the beginning of Section 3 we defined the tables $\mathcal{P}_{auth}$. These tables represent the information published by the authority and are supposedly subtables of the authority's table $\mathcal{E}$.

To verify that this is indeed the case and that the election outcome is correct, voters and auditors use additional information that is given by (i) the voting protocol itself and (ii) the voters' individual knowledge of their cast votes. To model this additional information, we denote by $\mathcal{P}_{con}$ the nonempty set containing all possible tables that satisfy publicly known constraints given by the protocol, such as the published tables' shape (determined when the number of votes is known) or the tally. We denote by $\mathcal{V}$ the set of voters, and for each voter v we model the knowledge available to her, e.g. her vote, by a smaller table R_v that indicates v's view of the election and hence contains v's *real* vote. These tables are not present in the protocol specification itself. We denote the combined view of all voters by $Views = \{R_v \mid v \in \mathcal{V}\}$. For each voter $v \in \mathcal{V}$, the table R_v has a column of sort V that in every row contains the identifier $v \in \mathcal{V}$. This sort is used to correctly assign v's ballots to v independently of any information published by the voting authority. Note that tables in $\mathcal{P}_{auth}$ do not have a column V and tables in *Views* do not contain information on printed, but unused ballots.

Eperio. In Eperio, $\mathcal{P}_{con}$ encodes the tally (denote by t, the announced outcome) and the constraints that all printed ballots must be represented by $P_1, \ldots, P_x$:

$$\mathcal{P}_{con} := Ind_t^{qu}(\pi_S \sigma_{M=1}) \cap \bigcap_{P \in \mathcal{P}_{auth}} \{T \in \mathcal{T}_\perp \mid P \doteq \pi_{\mathrm{sort}(P)} T\}.$$

For each voter $v \in \mathcal{V}$, R_v is a 4-column table of sort $\{V, U, M, S\}$, where column V contains v. The entries in column U of R_v contain the serial numbers and positions on the received ballot, the entries in column M contain the actual votes marked by v on her cast ballot, and column S contains the actual candidate names printed for the corresponding serial number and position on the ballot. We denote the set of these tables by $Views := \{R_v \mid v \in \mathcal{V}\}$.

Ideally, we can combine the voters' information *Views*. In this case, $\bigwedge^U Views$ is nonempty and contains all possible extensions of the provided information. Since all tables in *Views* are of the same sort and they do not contain $\perp$ values, the proof of Lemma 1 yields a table $\overline{V}$ such that $\bigwedge^U Views = [\overline{V}]^U$. Then the combined view of the voters is represented by the table $\overline{V} \in \mathcal{T}$, which is also the unique element in the set $\min \bigwedge^U Views$.

Otherwise there is conflicting information, and the set $\bigwedge^U Views$ is empty. This should be detected by the verification procedure.

4 VERIFIABILITY PROPERTIES

In Phase 3 of the Eperio protocol, voters and auditors conduct checks $\psi_1, \ldots, \psi_k \in \Psi$ on the sets of tables $\mathcal{P}_{auth}, \mathcal{P}_{con}$, and *Views*. These checks are intended to verify that the result published by the election authority corresponds to the voters' intentions, as in global verifiability [26]. We denote this integrity property by ϕ and will return to it in Section 4.2, Definition 4.

4.1 Properties of checks

We state now what we require for a general verification procedure, consisting of a set of properties $\Psi = \{\psi_1, \ldots, \psi_k\}$, to yield correct results. To begin with, a verification procedure is *sound* if all the checks in Ψ entail ϕ. For voting, soundness is essential as it amounts to the outcome's integrity and ensures that any cheating in the production of tables and ballots (represented by the tables in $\mathcal{P}_{auth}$ and $\mathcal{P}_{con}$) will be uncovered.

Completeness is the converse: ϕ entails all the properties in Ψ. In general, completeness may be too strong as an election outcome could correspond to the voters' intent even if errors occur. For example, an audit could detect errors in a batch of printed but unused ballots. The verification of this batch of printed ballots would fail, but the election's outcome may still be correct.

In practice, voting protocols need dispute resolution procedures [2] to achieve soundness and completeness in the case of *dishonest voters*, but this is out of this paper's scope. We assume that *Views* corresponds to the voters' intent.

Definition 3. *The verification of a property ϕ is* sound *with respect to Ψ if*

$$\bigwedge_{\psi_i \in \Psi} \psi_i(\mathcal{P}_{auth}, \mathcal{P}_{con}, Views) \Rightarrow \phi(\mathcal{P}_{auth}, \mathcal{P}_{con}, Views)\,. \tag{1}$$

The verification of ϕ is complete *with respect to Ψ if*

$$\bigwedge_{\psi_i \in \Psi} \psi_i(\mathcal{P}_{auth}, \mathcal{P}_{con}, Views) \Leftarrow \phi(\mathcal{P}_{auth}, \mathcal{P}_{con}, Views)\,. \tag{2}$$

Together, these requirements state that the conjunction of the properties $\psi_i \in \Psi$ is equivalent to ϕ. This admits a trivial solution where $\Psi = \{\phi\}$. In practice, since one also requires voter privacy, Ψ is instead decomposed into multiple verification checks, where each check only accesses a strict subset of the table's elements.

Definition 3 is specific to checks conducted on the sets of tables $\mathcal{P}_{auth}, \mathcal{P}_{con}$, and *Views* that describe an election. The definition is clearly generalizable to any property $\phi(\mathcal{S}_1, .., \mathcal{S}_k)$ and checks $\{\psi_i(\mathcal{S}_1, .., \mathcal{S}_k) \mid \psi_i \in \Psi\}$ conducted on arbitrary finite sets of tables $\mathcal{S}_1, .., \mathcal{S}_k \subseteq \mathcal{T}_\perp$ and for any property $\phi(T_1, .., T_k)$ and checks $\{\psi_i(T_1, .., T_k) \mid \psi_i \in \Psi\}$ conducted on arbitrary tables $T_1, .., T_k \in \mathcal{T}_\perp$.

Example 3. Consider the tables $T, S \in \mathcal{T}$ where, in contrast to Eperio, U values need not be unique.

$$T = \begin{array}{|cc|c|}\hline U & M & S \\ \hline a & 1 & no \\ a & 0 & yes \\ b & 1 & yes \\ b & 0 & no \\ \hline \end{array} \qquad S = \begin{array}{|cc|c|}\hline U & M & S \\ \hline a & 0 & no \\ a & 1 & yes \\ b & 1 & no \\ b & 0 & yes \\ \hline \end{array}$$

Let $\phi(T, S)$ be the property that the table T is equal to the table S. Let $\psi(T, S)$, $\psi'(T, S)$, and $\psi''(T, S)$ be the properties that T's projection on the first two columns, the first and last columns, and the last two columns, respectively, are equal to the corresponding projections on S. Then $\phi(T, S) \Rightarrow \psi(T, S) \wedge \psi'(T, S) \wedge \psi''(T, S)$, but the converse is false. The tables T and S are *not* equal, not even up to permutation of rows. So $\phi(T, S)$ is not satisfied. But *any two columns* of T are equal to the corresponding two columns of S, so $\psi(T, S)$, $\psi'(T, S)$, and $\psi''(T, S)$ are satisfied. Hence a verification procedure that aims to establish $\phi(T, S)$ by checking the properties $\psi(T, S)$, $\psi'(T, S)$, and $\psi''(T, S)$ would be complete, but not sound. $\triangle$

The following lemma states that, in contrast to the above example, if the tables contain a unique key, then it is possible to verify a global property by performing checks on subsets of columns.

Lemma 2. Let $\mathcal{R} \subseteq \mathcal{T}$ be a set of tables, defining a property, and $\{S_0, S_1, S_2, ..., S_k\}$ a subsort of sort(R) for all $R \in \mathcal{R}$. Let $T \in \mathcal{T}$ be a table such that $T \in \mathbin{⩓}^{\{S_0\}}\{\pi_{S_0,S_i} R \mid R \in \mathcal{R}\}$ for all $i \in \{1, \ldots, k\}$. Then

$$T \in \mathbin{⩓}^{\{S_0\}}\{\pi_{S_0,S_1,..,S_k} R \mid R \in \mathcal{R}\}.$$

This lemma will be crucial for the proof of correctness of Eperio's verification procedure.

4.2 Security goals

Observe that the sets of tables $\mathcal{P}_{auth}, \mathcal{P}_{con}$, and *Views* may result in different, potentially conflicting views of the overall election data. A voting protocol must therefore ensure that:

(1) the voters' joint view of the election can be combined (in the sense of Section 3.2) with the published data and
(2) every election table that combines the public data and the voters' joint views yields the same election outcome.

We formalize these two conditions as follows.

Definition 4. *Given a non-empty sort Key of unique keys, $\mathcal{P}_{auth}$, Views, and $\mathcal{P}_{con}$ as in Section 3, let $C = \mathbin{⩓}^{Key}(\mathcal{P}_{auth} \cup Views) \cap \mathcal{P}_{con}$.*

We say that an election's outcome *tal* corresponds to the voters' intent*, and denote this property by $\phi_{Key,\mathrm{tal}}(\mathcal{P}_{auth}, \mathcal{P}_{con}, Views)$, if the following two conditions are satisfied.*

(1) $C \neq \emptyset$ *and*
(2) $\forall T \in C, \forall T' \in \min \mathbin{⩓}^{Key} Views\colon \mathrm{tal}(T) = \mathrm{tal}(T')$,

where tal *denotes the tally function.*

For clarity, we will henceforth omit subscripts and arguments and write ϕ.

As a side remark, Definition 4 also illustrates how our operators provide a concise way to specify (and later reason about) relevant properties of voting, formulated on tables.

The property ϕ covers the central integrity requirement (i.e., the election's result reflects the intent of the electorate) for voting protocols, a point we discuss further in Section 8. Establishing that the verification checks made in a voting protocol imply ϕ hence corresponds to ensuring end-to-end verifiability. Note that this definition can still be flexibly adapted to the specifics of different protocols as it is parametrized by the tally function *tal* and the protocol-specific tables $\mathcal{P}_{auth}, \mathcal{P}_{con}$, *Views*, and the unique keys *Key*.

4.3 Verifiability for Eperio

4.3.1 Overview. In [22], to show that the verification procedure guarantees that the election is counted-as-intended, the authors prove that each step of the audit is sound and complete.

In particular, they compute the probability that each check succeeds in detecting a malicious execution. For example, for the verification step where voters check their receipts using columns U and M, they compute the probability of detecting ballot receipts that have been modified. However, no other type of malicious behaviour is considered. In other words, soundness is shown for each single verification step, but not for the entire verification procedure.

In contrast to the original analysis [22], we analyze the soundness of the overall verification procedure, where we perform a possibilistic rather than a probabilistic analysis. Thus we assume that if a mistake can be detected (with some probability), then this will be the case.

As we shall see, the use of our methodology shows that to prove that Eperio's checks are sound, additional assumptions (or alternatively additional checks) are required. We shall provide such assumptions, which can be easily fulfilled. That weaknesses for this protocol exist, despite the proofs in [22], underscores the importance of formalizing and analyzing table protocols *before* conducting a probabilistic analysis.

4.3.2 Modeling the Verification Steps. There are three verification steps in Phase 3 in Eperio. We explain each step, formulate the corresponding properties ψ_i, and afterwards analyze how they relate to ϕ. In addition to these steps, the voting protocol limits what the voting authority can do. The general shape of the tables published by the authority is fixed and will be verified by auditors. Moreover, column M is known for every table in $\mathcal{P}_{auth}$, and there are tables $P, Q \in \mathcal{P}_{auth}$ such that column U is known in P and column S is known in Q. The auditors verify that all the known columns of sorts $\{U, M\}$ and $\{M, S\}$ are equal in all tables and that the known values in column U are unique. We represent the verification of all these properties by ψ_0 and define

$$\psi_0 := \mathbin{⩓}^{\{U\}}\mathcal{P}_{auth} \cap \bigcap_{P \in \mathcal{P}_{auth}} shape_P \neq \emptyset.$$

Verification that votes were recorded as cast. Each ballot contains a serial number that allows a voter to verify a specific range of entries in column M. To verify that their votes were recorded as cast, each voter $v \in \mathcal{V}$ must check that the marks on her ballot for her serial numbers are present in the information $\mathcal{P}_{auth}$ revealed by the authority for the same serial numbers.

$$\psi_1 := \bigwedge_{v \in \mathcal{V}} \bigwedge_{\substack{P \in \mathcal{P}_{auth} \\ \pi_{U,M} P \in \mathcal{T}}} (\pi_{U,M} R_v \subseteq^{\#} \pi_{U,M} P).$$

The conjunction of these properties is weaker than the requirement that the voters' combined view is consistent with the published tables, which is what Definition 4 requires. In particular, ψ_1 neither implies that all serial numbers are unique nor does it prevent ballot stuffing. We will return to these points in Section 4.4.

Verification that ballots were printed correctly. Every voter has the option of keeping a ballot for print auditing. The individual print-audit checks verify that the voter's view of a print-audited ballot matches the information in $\mathcal{P}_{auth}$ published by the authority. Each voter $v \in \mathcal{V}$ possessing a print-audit ballot checks that $\pi_{U,M,S}\sigma_{M=-1} R_v \subseteq^{\#} \sigma_{M=-1} P$, for all $P \in \mathcal{P}_{auth}$. Since the election authority commits to the columns *before* knowing which ballots are print audited, we will make the assumption that these random audits force the authority to print correctly (i.e., as observed by the voters) also the non-print-audited ballots. We denote this property by $\psi_{\text{print-audit}}$ and define it as a consistency check of the voters' combined view of the U and S columns matched against the public

information (see ψ_0):

$$\begin{aligned}\psi_{\text{print-audit}} \quad &:= \quad \textcircled{\wedge}^{\{U\}}\{\pi_{U,S}(R) \mid R \in \mathit{Views}\} \\ &\cap \quad \textcircled{\wedge}^{\{U\}}\mathcal{P}_{auth} \cap \bigcap_{P\in\mathcal{P}_{auth}} \mathit{shape}_P \neq \emptyset.\end{aligned}$$

Verification of the tally. Let t be the announced tally, given as a table of sort S. Each auditor and interested voter checks that for all P in $\mathcal{P}_{auth}$ the table t is in the set $\mathit{Ind}_P^{qu}(\{\pi_S\sigma_{M=1}\})$. Thus we define ψ_{tally} as

$$\psi_{\text{tally}} := t \in \bigcap_{P\in\mathcal{P}_{auth}} \mathit{Ind}_P^{qu}(\{\pi_S\sigma_{M=1}\}).$$

4.4 Soundness and Completeness for Eperio

We now consider the relationship between $\psi = \psi_0 \wedge \psi_1 \wedge \psi_{\text{tally}} \wedge \psi_{\text{print-audit}}$ and ϕ.

4.4.1 Missing Checks in Eperio. As stated in Section 4.3.2, duplicate serial numbers and stuffed ballots may fail to be detected. We demonstrate this with a simple example. Let $\mathcal{E}$ be the authority's voting table and R be the combined view of the voters with the following values.

$$\mathcal{E} = \begin{array}{|ccc|}\hline U & M & S \\ \hline 1.1 & 0 & x \\ 1.2 & 1 & y \\ 2.1 & 0 & y \\ 2.2 & 1 & x \\ 3.1 & 0 & y \\ 3.2 & 1 & x \\ \hline\end{array} \qquad R = \begin{array}{|cccc|}\hline V & U & M & S \\ \hline a & 1.1 & 0 & x \\ a & 1.2 & 1 & y \\ b & 2.1 & 0 & y \\ b & 2.2 & 1 & x \\ c & 1.1 & 0 & x \\ c & 1.2 & 1 & y \\ \hline\end{array}$$

Observe that voters a and c have been given the same ballot number 1. Suppose the voting authority publishes $\mathcal{P}_{auth}$ and $\mathcal{P}_{con}$ by following the protocol correctly, thus satisfying ψ_0. Suppose each voter knows only what was written on her ballot and who she voted for, that is for each voter $v \in \{a, b, c\}$, $R_v = \sigma_{V=v}(R)$. Then ψ_1 is satisfied since the individual checks of all three voters are consistent with $\pi_{U,M}P$ for $P \in \mathcal{P}_{auth}$, even though voters a and c are both mapped to the same ballot. Moreover, no voter has voted with ballot 3, yet a vote for candidate y is recorded. The properties ψ_1, ψ_{tally}, and $\psi_{\text{print-audit}}$ are easily seen to be satisfied.

However, ϕ is not satisfied since $\textcircled{\wedge}^{\{U\}}(\mathcal{P}_{auth} \cup \{R_a, R_b, R_c\}) = \emptyset$. This is because $R_a \,\textcircled{\wedge}^{\{U\}}\, R_c = \emptyset$, since there is no extension of these two tables with unique keys 1.1 and 1.2 and different values for these keys in column V.

Note that even if voter c had not voted (remove voter c's rows), all checks would be satisfied, but ϕ would still not be. This is because of the second condition of ϕ, where the voter's combined view yields a different tally than what is announced by the authority.

We conclude that the checks in Eperio are insufficient to guarantee that the election is counted as intended. For example, assume that some dishonest officials at a polling station, colluding with the authority, want to decrease the votes for targeted candidates. If they expect two voters to vote for the same candidate (e.g., based on their backgrounds or party memberships), they can give each a copy of the same paper ballot. If both voters vote for the same candidate, all checks succeed, but only one vote is counted.

4.4.2 Soundness of Eperio under additional assumptions. To determine whether there are further problems, we now assume that all ballots have unique serial numbers and we call this property

$$\psi_{\text{unique}} := \textcircled{\wedge}^{\{U\}}\{\pi_{V,U}R \mid R \in \mathit{Views}\} \neq \emptyset.$$

We also assume that no ballot stuffing occurs. Recalling that $\overline{V} = \min \textcircled{\wedge}^{\{U\}}\mathit{Views}$, we set

$$\psi_{\text{no-stuffing}} \quad := \quad \forall P \in \mathcal{P}_{auth}\colon \#\sigma_{M=1}\pi_M P = \#\sigma_{M=1}\pi_M\overline{V}.$$

These additional assumptions should be replaced by additional checks in practice, for example by publicly tracking all used ballot identifiers to avoid collisions.

To show soundness of the fixed version of Eperio, we now want to show that

$$\psi_0 \wedge \psi_1 \wedge \psi_{\text{tally}} \wedge \psi_{\text{print-audit}} \wedge \psi_{\text{unique}} \wedge \psi_{\text{no-stuffing}} \Rightarrow \phi.$$

Thus, under the assumption that no ballot stuffing occurs and all ballots are printed with unique serial numbers, the Eperio verification procedure is sound (in the possibilistic setting).

Let $\mathcal{P} := (\textcircled{\wedge}^{\{U\}}\mathcal{P}_{auth}) \cap (\cap_{P\in\mathcal{P}_{auth}}\mathit{shape}_P)$, which is not empty by ψ_0. The check $\psi_{\text{print-audit}}$ gives us a table $T \in \mathcal{T}$ such that

$$T \in (\textcircled{\wedge}^{\{U\}}\{\pi_{U,S}R \mid R \in \mathit{Views}\}) \cap \mathcal{P}.$$

Since the columns of sort $\{U, M\}$ in $\mathcal{P}$ must be consistent with those in each table in $P \in \mathcal{P}_{auth}$, the check ψ_1 implies that

$$\bigwedge_{P\in\mathcal{P}} (\pi_{U,M}R_v \subseteq^{\#} \pi_{U,M}P).$$

In particular, this also holds for $T \in \mathcal{P}$ so that

$$T \in \textcircled{\wedge}^{\{U\}}\{\pi_{U,M}R \mid R \in \mathit{Views}\}.$$

By Lemma 2 we conclude that

$$T \in \textcircled{\wedge}^{\{U\}}\{\pi_{U,M,S}R \mid R \in \mathit{Views}\}.$$

ψ_{unique} guarantees that $\min \textcircled{\wedge}^{\{U\}}\{\pi_{V,U}R \mid R \in \mathit{Views}\} = [Z]^{\{U\}}$ for some table Z with unique elements of sort U. Since T does not have a V column, we may extend it horizontally: for the tuples t in T whose primary key of sort Key is in Z, we set $t(V)$ to be the corresponding V value in that table, and by the uniqueness of U values in Z, this is well defined. For the other tuples t in T, we set $t(V)$ to be any arbitrary value. This gives us a table $E \in \textcircled{\wedge}^{\{U\}}\mathit{Views} \cap \textcircled{\wedge}^{\{U\}}\mathcal{P}_{auth}$ that has the same number of rows as T (hence of the tables $P \in \mathcal{P}_{auth}$), so that for all $P \in \mathcal{P}_{auth}$, $\pi_{\text{sort}(P)}E \triangleq P$. Hence

$$\begin{aligned}E \in \mathcal{C}' = &\textcircled{\wedge}^{\{U\}}(\mathit{Views}) \cap \textcircled{\wedge}^{\{U\}}(\mathcal{P}_{auth}) \\ &\cap \bigcap_{P\in\mathcal{P}_{auth}} \{T \in \mathcal{T} \mid P \triangleq \pi_{\text{sort}(P)}T\} \neq \emptyset.\end{aligned}$$

In particular, since there is at least one table in $\mathcal{P}_{auth}$ that reveals its columns M and S,

$$\forall T \in \mathcal{C}', P \in \mathcal{P}_{auth}\colon \pi_{M,S}(T) \triangleq \pi_{M,S}P.$$

Moreover, since by ψ_{tally} we have $t \in \mathit{Ind}_P^{qu}(\{\pi_S\sigma_{M=1}\})$ for all $P \in \mathcal{P}_{auth}$, we conclude that $t \in \mathit{Ind}_T^{qu}(\{\pi_S\sigma_{M=1}\})$ for all $T \in \mathcal{C}'$ so that

$$\mathcal{C} = \mathcal{C}' \cap \mathit{Ind}_t^{qu}(\{\pi_S\sigma_{M=1}\}) \neq \emptyset.$$

Now consider the second predicate of ϕ. The above implies that $\textcircled{\wedge}^{\{U\}}\mathit{Views}$ is not empty, so $\overline{V} = \min \textcircled{\wedge}^{\{U\}}\mathit{Views}$ is well defined

and has unique entries of sort U. Given that for all $T \in \mathcal{C}$ and $P \in \mathcal{P}_{auth}$ we have $\pi_{\{M,S\}} T \triangleq P$, the property $\psi_{\text{no-stuffing}}$ implies that for all $T \in \mathcal{C}$,

$$\#\sigma_{M=1}\pi_M T = \#\sigma_{M=1}\pi_M \overline{V}. \tag{3}$$

Since $\mathcal{C}$ is defined as an extension of the tables in *Views*, for any $T \in \mathcal{C}$, $\overline{V} \subseteq^{\#} T$. Combining this with (3) we indeed obtain that for all such T,

$$\pi_S \sigma_{M=1} T = \pi_S \sigma_{M=1} \overline{V}.$$

4.4.3 Completeness. It is straightforward to show completeness using contraposition. That is, if ψ_i does not hold, for some $i \in \{0, 1, \text{tally}, \text{print-audit}, \text{unique}, \text{no-stuffing}\}$, then neither does ϕ. See Appendix C for the proof.

5 PRIVACY

In this section we will analyze the privacy of table-based verification procedures to ensure that no information is leaked from the published tables $\mathcal{P}_{auth}$ and $\mathcal{P}_{con}$. That is, any assignment of votes to voters should be indistinguishable from any other assignment, based on the tables $\mathcal{P}_{auth}$ and $\mathcal{P}_{con}$.

In the following, we assume that the verification itself is successful. In particular, we assume that:

(i) Each table $P \in \mathcal{P}_{auth}$ is a subtable of $\mathcal{E}$. Hence if Q is the set of queries used for the audits, $\mathcal{P}_{auth} = \{q(\mathcal{E}) \mid q \in Q\}$.
(ii) $\bigwedge^{Key}(\mathcal{E} \cup \mathit{Views}) \cap \mathcal{P}_{con} \neq \emptyset$.

We justify this assumption by noting that a cheating election authority can easily leak information about the voters, so typically we cannot achieve privacy when the election authority is dishonest. Observe that assumptions (i) and (ii) together imply that $\bigwedge^{Key}(\mathcal{P}_{auth} \cup \mathit{Views}) \cap \mathcal{P}_{con} \neq \emptyset$.

Note that our privacy analysis only concerns verification checks, and not the entire protocol execution. This means that most existing privacy definitions (see e.g., [6]) are not directly applicable in our context: they are either defined in terms of a cryptographic game invoking various algorithms and oracles, or the indistinguishability of different executions of the entire voting protocol. In contrast, here we deal with different views on a single table, representing just the verification checks.

5.1 Defining vote privacy for table checks

We shall now define vote privacy, taking inspiration from symbolic vote privacy definitions [19], where privacy is defined as the inability to distinguish two instances of the protocol where two voters swap votes. Namely, we say that a verification procedure ensures voter privacy if any permutation of the voter's *private votes* (i.e., ballots that are counted, rather than, e.g., print audited) is consistent with the published data.

Let $\tilde{S}_{\mathcal{V}}$ be a group that contains all permutations of voters that have cast a private ballot. For each element $g \in \tilde{S}_{\mathcal{V}}$ there is a function $\tau \in S_{\mathcal{V}}$ on the tables in *Views*, denoted $\tau(\mathit{Views}) := \{\tau(R_v) \mid R_v \in \mathit{Views}\}$, that transforms the tables in *Views* by swapping the votes of voters as specified by g. The formal definition of $S_{\mathcal{V}}$ depends on the voting protocol and the election table, as discussed below. Given $S_{\mathcal{V}}$, we can define vote privacy for table-based verification checks.

Definition 5. *A verification procedure ensures vote privacy if given the unique key Key, the public data $\mathcal{P}_{auth}$ and $\mathcal{P}_{con}$, and the voters' views Views $= \{R_v \mid v \in \mathcal{V}\}$, then for* all $\tau \in S_{\mathcal{V}}$,

$$\bigwedge^{Key}(\mathcal{P}_{auth} \cup \tau(\mathit{Views})) \cap \mathcal{P}_{con} \neq \emptyset.$$

To define how a function τ in $S_{\mathcal{V}}$ transforms the tables in *Views*, it suffices to define how τ transforms a table $\overline{C} \in \min \bigwedge^{Key}(\mathcal{E} \cup \mathit{Views}) \cap \mathcal{P}_{con}$, given by assumption (ii). By definition of the merge operator, such a table contains each voter's view, given by the table $R_v = \sigma_{V=v}\overline{C}$ (for any choice of $\overline{C}$).

We partition the sort of the voters' tables into two subsorts: *Fixed* and *Choice*. *Fixed* contains the column names for data that is fixed for each voter (e.g. the voter's identity, her ballot number, etc.). Typically, *Fixed* will contain V and be a superset of *Key*. *Choice* contains column names for data that indicates the voters' choices (e.g. the chosen candidates) and to ensure the voter's privacy, this data must not all be associated with the data in *Fixed* as this would reveal the voters' votes. For a voter $v \in \mathcal{V}$, let vote_v be the query that returns v's vote. Finally we denote by σ_{pub} the query that selects publicly audited votes, if there are any.

Concretely $\tau \in S_{\mathcal{V}}$ is then a function that permutes the subtables $\{\mathit{vote}_v(\overline{C}) \mid v \in \mathcal{V}\}$. To define $S_{\mathcal{V}}$ it suffices to define how a function $\tau_{vv'} \in S_{\mathcal{V}}$ swaps the votes of v and v', for any two voters v and v' that have cast a vote. Any permutation of votes is then obtained by a sequence of vote swaps.

Definition 6. *Let T be a table such that* $\text{sort}(T) = \text{sort}(\overline{C})$ *and $\pi_V T = \pi_V \overline{C}$. For $v, v' \in \pi_V T$, we define $\tau_{vv'}(T)$ to be the table T' such that:*

(1) *For all $v'' \in \mathcal{V} \setminus \{v, v'\}$: $\sigma_{V=v''}(T') = \sigma_{V=v''}(T)$*
(2) *$\sigma_{pub}(T') = \sigma_{pub}(T)$*
(3) *$\pi_{Fixed} T' = \pi_{Fixed} T$*
(4) *$\mathit{vote}_v(T') = \mathit{vote}_{v'}(T)$ and $\mathit{vote}_{v'}(T') = \mathit{vote}_v(T)$*

Conditions (1) and (2) fix all tuples of public ballots and of voters other than v and v'. Condition (3) fixes all tuples in T except for the tuples in column *Choice*, thus allowing the tuples in *Choice* for voters v and v' to be permuted. In particular this implies that $\tau \in S_{\mathcal{V}}$ is defined as a permutation of the tuples of sort *Choice* in $\overline{C}$. Condition (4) requires that the votes of v and v' are swapped. These four conditions do not uniquely determine the table T' in general. Thus for each pair of voters $\{v, v'\}$, there may be several table functions that swap their votes. We define $S_{\mathcal{V}}$ by requiring that for any transposition of voters in $\tilde{S}_{\mathcal{V}}$ that have cast a vote, there is a $\tau_{vv'} \in S_{\mathcal{V}}$ satisfying the Conditions (1)–(4) and that the elements of $S_{\mathcal{V}}$ form a group under composition.

Eperio. In Eperio, cast votes are determined by the tuples of sort $\{M, S\}$ that have a 0 or a 1 in column M. Hence

$$\mathit{Fixed} = \{V, U\} \text{ and } \mathit{Choice} = \{M, S\}.$$

Similarly, for a voter $v \in \mathcal{V}$,

$$\mathit{vote}_v = \sigma_{M \in \{0,1\}} \pi_{MS} \sigma_{V=v}$$

is the function that returns v's choice.

Finally print-audited ballots are those marked with -1 so that $\sigma_{pub} = \sigma_{M=-1}$.

Following Definition 6, for two voters v and v', $\tau_{vv'} \in S_{\mathcal{V}}$ transforms the tables in *Views* by swapping *all* the tuples of columns M and S of the subtable $\sigma_{V=v}(\bar{C})$ with *all* those of the subtable $\sigma_{V=v'}(\bar{C})$ so as to exchange the marked candidates of the two voters. For example, the following two tables illustrate a vote swap in Eperio's protocol.

Example 4.

$$\bar{C} = \begin{array}{|cccc|}\hline V & U & M & S \\ \hline v & H.1 & 1 & Alice \\ v & H.2 & 0 & Bob \\ v & H.3 & 0 & Charlie \\ v' & N.1 & 0 & Bob \\ v' & N.2 & 1 & Charlie \\ v' & N.3 & 0 & Alice \\ \hline \end{array}, \quad \tau_{vv'}(\bar{C}) = \begin{array}{|cccc|}\hline V & U & M & S \\ \hline v & H.1 & 1 & Charlie \\ v & H.2 & 0 & Bob \\ v & H.3 & 0 & Alice \\ v' & N.1 & 0 & Bob \\ v' & N.2 & 1 & Alice \\ v' & N.3 & 0 & Charlie \\ \hline \end{array}.$$

△

Proving Privacy. To analyze privacy in the sense of Definition 5, it suffices to determine how the elements of sort *Choice* may be permuted while remaining consistent with both the information $\mathcal{P}_{auth}$ revealed by the authority for auditing purposes and the publicly known constraints $\mathcal{P}_{con}$. The information revealed during the verification procedure is given by the set of queries Q used for the audits, i.e., $\mathcal{P}_{auth} = \{q(\mathcal{E}) \mid q \in Q\}$. Hence ensuring that

$$\forall \tau \in S_{\mathcal{V}} : \textstyle\bigwedge^{Key}(\mathcal{P}_{auth} \cup \tau(Views)) \cap \mathcal{P}_{con} \neq \emptyset,$$

corresponds to verifying that the elements of sort *Choice* can be permuted as specified by $S_{\mathcal{V}}$ without being detected by any of the queries in Q or excluded by the constraints in $\mathcal{P}_{con}$. In terms of indistinguishability sets, any permutation of tuples of sort *Choice* in $\bar{C}$ as specified by any element of $S_{\mathcal{V}}$ should be in the indistinguishability set $Ind_{\bar{C}}(Q, \mathcal{P}_{con})$ associated to Q. In particular this means that for all $\tau \in S_{\mathcal{V}}$, that $\tau(\bar{C}) \in \bigwedge^{Key}(\mathcal{P}_{auth} \cup \tau(Views)) \cap \mathcal{P}_{con}$.

5.2 G-indistinguishability

To support our privacy analysis, we formalize a novel quantitative privacy notion in terms of indistinguishability sets (Definition 1). This notion is easily applied to the table $\bar{C}$ and the protocol queries Q used during the audit. It accounts for information that has been revealed and expresses how tables may be permuted while remaining consistent with this revealed data.

Definition of G-indistinguishability. Let $P_T(N)$ be the set of functions that permute the elements of $T \in \mathcal{T}$ within each of the columns in $N \subseteq \text{sort}(T)$. More precisely, $P_T(N)$ is the symmetric group on $\bigcap_{n \in N}[\pi_n T] \cap [\pi_{\text{sort}(T)\setminus N}] \cap shape_T$.

Definition 7. *Let Q be a subset of projection and selection queries.*

- *The elements of sort N of a table T are said to be* completely indistinguishable *under the queries in Q if for all $p \in P_T(N)$ it holds that $p(T) \in Ind_T(Q, \{shape_T\})$.*
- *The elements are said to be* G-indistinguishable *under the queries in Q for G a subgroup of $P_T(N)$ if $\forall\, p \in G\colon p(T) \in Ind_T(Q, \{shape_T\})$.*

Example 5. Consider the table Y in Example 2.(1) and the set of queries $Q = \{\pi_{\{U,M\}}, \pi_{\{M,S\}}\}$. None of Y's subsorts have completely indistinguishable elements under the set of queries Q, by Definition 7. However, the elements of sort $\{U\}$ are G-indistinguishable for the group G (of order 2) generated by the transposition that swaps the top $H.1$ with $N.1$ in column U of Y. △

Relationship to privacy. Observe that we have defined $S_{\mathcal{V}}$ as a subgroup of $P_{\bar{C}}(Choice)$, for a fixed $\bar{C} \in \min \bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con}$. The next theorem gives a criterion for the vote privacy (Definition 5) of a verification procedure based on $S_{\mathcal{V}}$-indistinguishability.

Theorem 1. Suppose that $\mathcal{P}_{auth} = \{q(\mathcal{E}) \mid q \in Q\}$ and $(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con}) \neq \emptyset$. Then a protocol's verification procedure guarantees vote privacy as in Definition 5 provided that for any $\bar{C} \in \min(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con})$ the tuples of sort *Choice* of $\bar{C}$ are $S_{\mathcal{V}}$-indistinguishable under the queries in Q and that $\tau(\bar{C}) \in \mathcal{P}_{con}$ for all $\tau \in S_{\mathcal{V}}$.

Fix an arbitrary $\bar{C} \in \min(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con})$. Clearly if the tuples of sort *Choice* of $\bar{C}$ are G-indistinguishable under the queries in Q for a group G that contains $S_{\mathcal{V}}$ as a subgroup, then these tuples are also $S_{\mathcal{V}}$-indistinguishable under the queries in Q. Hence to prove that a protocol ensures vote privacy, we apply the following steps:

(1) Consider the group $S_{\mathcal{V}}$ described in Definition 6.
(2) Compute the group G for which tuples of sort *Choice* of $\bar{C}$ are G-indistinguishable under the query set Q.
(3) Verify that $S_{\mathcal{V}}$ is indeed a subgroup of G.
(4) Verify that for any $\bar{C} \in \min(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con})$ then $\tau(\bar{C}) \in \mathcal{P}_{con}$ for all $\tau \in S_{\mathcal{V}}$.

We now apply these steps to the Eperio protocol. First, as described in Section 5.1, the group $S_{\mathcal{V}}$ is generated by the transpositions $\{\tau_{vv'} \mid v, v' \in \mathcal{V}\}$. A transposition $\tau_{vv'}$ swaps *all* the tuples of columns M and S belonging to a non-public vote of the subtable $\sigma_{V=v}(\bar{C})$ with *all* those of the subtable $\sigma_{V=v'}(\bar{C})$, exchanging the votes of v and v'. In particular $\tau_{vv'}$ swaps the candidate marked by v and the one marked by v'. As a second step, the following lemma gives us the group for which elements of sort *Choice* $= \{M, S\}$ are indistinguishable.

Lemma 3. In Eperio, the elements of sort *Choice* $= \{M, S\}$ in $\bar{C}$ are $G_{0,1}$-indistinguishable under the query set $Q = \{\pi_U, \pi_{U,M}, \sigma_{M=-1}, \pi_{M,S}\}$, where $G_{0,1}$ is the product of two groups that permute tuples of sort $\{M, S\}$. The first group arbitrarily permutes elements of sort $\{M, S\}$ that belong to a tuple whose value of sort M is 0, the second group those whose value of sort M is 1.

Recall that during the Eperio verification procedure analyzed in Section 4.3, the auditors receive table data that amounts exactly to the queries in the set Q specified in Lemma 3.

Observe that the group $G_{0,1}$ contains permutations of elements that may result in invalid ballots. For example, consider the table $\overline{V}$ from Example 4. The element $g \in G_{0,1}$ that only swaps $\langle M : 1, S : Alice\rangle$ with $\langle M : 1, S : Charlie\rangle$ yields a table where voter v has a candidate list that contains *Alice* twice, and v' a list containing *Charlie* twice. However, for the third step, if we require that all voters make the same number of marks (we will return to this later) it is clear that $G_{0,1}$ does contain $S_{\mathcal{V}}$ as a subgroup.

As a fourth step, recall that

$$\mathcal{P}_{con} := Ind_t^{qu}(\pi_S \sigma_{M=1}) \cap \bigcap_{P \in \mathcal{P}_{auth}} \{T \in \mathcal{T}_\perp \mid P \doteq \pi_{\mathrm{sort}(P)} T\}.$$

For any $\tau \in S_\mathcal{V}$, for any table $\bar{C} \in \mathcal{P}_{con}$, $\pi_s \sigma_{M=1}(\tau(\bar{C})) = \pi_s \sigma_{M=1} \bar{C}$ as τ only permutes marked candidates. Similarly $\pi_{U,S}(\tau(\bar{C})) = \pi_{U,S}\bar{C}$ and by definition $\pi_{U,M}(\tau(\bar{C})) = \pi_{U,M}\bar{C}$. Hence $\tau(\bar{C}) \in \mathcal{P}_{con}$ for all $\tau \in S_\mathcal{V}$. By Theorem 1, we obtain the following corollary.

Corollary 1. *The modified Eperio verification procedure ensures vote privacy according to Definition 5.*

Even when a protocol fails to guarantee vote privacy, G-indistinguishability for a group G that does not contain $S_\mathcal{V}$ still provides useful information. The group structure indicates where and to what extent privacy is preserved. We will compare this notion with other database privacy notions, such as k-anonymity, in Section 8.3.

Example 6. While not specified in [22], it is implicitly assumed that all voters mark the same number of candidates. In the case where the number of marks may differ, privacy is breached. For example, the following two tables show the action of $\tau_{vv'}$ on $\bar{C}$ for two voters v and v', where v has marked two candidates.

$$\bar{C} = \begin{array}{|cccc|}\hline V & U & M & S \\ \hline v & H.1 & 1 & Alice \\ v & H.2 & 0 & Bob \\ v & H.3 & 1 & Charlie \\ v' & N.1 & 0 & Bob \\ v' & N.2 & 1 & Charlie \\ v' & N.3 & 0 & Alice \\ \hline \end{array}, \quad \tau_{vv'}(\bar{C}) = \begin{array}{|cccc|}\hline V & U & M & S \\ \hline v & H.1 & 0 & Alice \\ v & H.2 & 0 & Bob \\ v & H.3 & 1 & Charlie \\ v' & N.1 & 0 & Bob \\ v' & N.2 & 1 & Charlie \\ v' & N.3 & 1 & Alice \\ \hline \end{array}.$$

In this case, the group $G_{0,1}$ computed in Lemma 3 does *not* contain $S_\mathcal{V}$ as a subgroup. Indeed, the element $\tau_{vv'}$ illustrated in the tables swaps, among others, the marked tuple $\langle M : 1, S : Alice \rangle$ with an unmarked tuple $\langle M : 0, S : Alice \rangle$.

When elements of $S_\mathcal{V}$ do not belong to $G_{0,1}$, these elements provide information on how vote privacy is breached. In this simple example, the swap of $\langle M : 1, S : Alice \rangle$ with $\langle M : 0, S : Alice \rangle$ does not belong to $G_{0,1}$. This shows that given the columns of sort $\{U, M\}$ and the final tally containing one vote for Charlie and two for Alice, one can indeed infer the vote $\langle M : 1, S : Alice \rangle$ made by voter v. △

6 SCANTEGRITY

In this section we illustrate our methodology on Scantegrity [12] with similar results. Like Eperio, Scantegrity uses table permutations to enable end-to-end verifiability while preserving privacy.

Scantegrity ballots are identified by unique serial numbers. As in Eperio, each ballot contains a list of candidates and a markable region (e.g. a bubble) for each candidate. Furthermore each candidate is associated to a code letter, in a random order for each ballot. To vote, voters mark a bubble and keep a receipt that consists of a serial number and the letter code corresponding to the chosen candidate, see Figure 4. We therefore model the voter's view as a table of sort $\{V, ID, L, M, S\}$. Column V contains the voter's identity and column ID specifies the ballot serial number. Column L contains the letters associated to each markable region. Finally, elements in column M indicate whether a region has been marked or not and those in column S specify the candidate corresponding to the region.

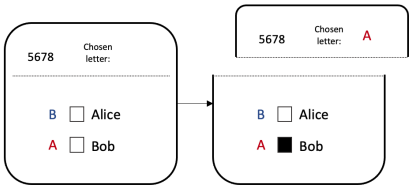

Figure 4: Scantegrity's ballot with receipt.

As in Eperio, the verification procedure is conducted on a table containing all scanned ballots. We hence model the authority's knowledge of the election as a table $\mathcal{E}$, which contains all Scantegrity serial numbers, of sort $\{ID, L, M, S\}$. A ballot may correspond to multiple rows, with the same serial number, so the unique identifier is the sort $Key = \{ID, L\}$. Observe that if we consider this pair as a unique column (i.e., we name $U = \{ID, L\}$), then we obtain exactly the same table sort as in Eperio.

Election procedure. Before the election (the column M is still empty), the election authority EA generates a so-called *switchboard*, which is a collection of circuits that link the markable regions on the ballots to their corresponding candidate names in permuted order. In particular the switchboard contains a link for each row of columns $\{ID, L, M\}$ to the corresponding candidate name in a permuted copy of columns $\{M, S\}$. The link between the code letter and the result has an intermediate permutation of column M that is used for auditing, see Figure 5.

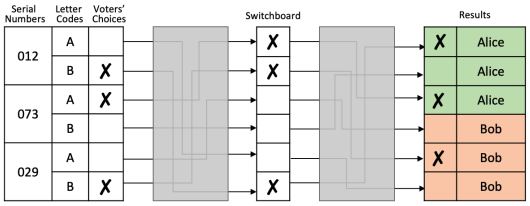

Figure 5: The switchboard after votes are cast.

The EA commits to the ballots it has generated and to the connections on the switchboard. Auditors then randomly choose half the ballots to be revealed and use these revealed ballots to check the associations on the switchboard. In particular, auditors check that the serial number and code-letter pair is linked to the correct candidate for each print-audited ballot. These ballots are later destroyed.

After votes are cast, column M is completed and the marks are sent through the switchboard. The voters then perform another audit. They challenge the EA to open one half of the switchboard (i.e. the link between code letters and marks) or the other (the link between marks and candidates).

We model this by having the EA publish either a table P_L that reveals the links between ballot serial numbers, code letters, and marks (hence a table of sort $\{ID, L, M, S\}$ with unknown values for the S column) or a table P_S that reveals the association between marks and candidates (hence a table of sort $\{ID, L, M, S\}$ with unknown values for the $\{ID, L\}$ tuples). Since the authority does not

know a priori which side will be chosen, we model $\mathcal{P}_{auth} = \{P_L, P_S\}$. The authors of [12] mention that to increase the audit's statistical certainty, multiple copies of the switchboard (each with randomly permuted links) may be used, giving rise to a set $\mathcal{P}_{auth}$ with multiple permuted copies of P_L and P_S. Observe that this corresponds exactly to the tables in $\mathcal{P}_{auth}$ in our model of Eperio.

Similarly, the tally function is $tal := \sigma_{M=marked}\pi_S$. Let t be the announced tally. The publicly known contraints are given by

$$\mathcal{P}_{con} := Ind_t^{qu}(\pi_S\sigma_{M=marked}) \cap \bigcap_{P\in\mathcal{P}_{auth}} \{T \in \mathcal{T}_\perp \mid P \doteq \pi_{\mathrm{sort}(P)}T\}.$$

Moreover, it is straightforward to see that the verification checks are analogous to those carried out in Eperio. Indeed the first audit, which checks the associations on the switchboard before marks are recorded, corresponds to the verification check

$$\psi_{\text{print-audit}} := \textcircled{\wedge}^{\{ID,L\}}\{\pi_{ID,L,S}R \mid R \in Views\} \cap \textcircled{\wedge}^{\{ID,L\}}\mathcal{P}_{auth} \cap \bigcap_{P\in\mathcal{P}_{auth}} shape_P \neq \emptyset.$$

The verification check that reveals the left part of the switchboard verifies that the voters' receipts are correctly recorded. This corresponds to the check

$$\psi_1 := \bigwedge_{v\in\mathcal{V}} (\pi_{ID,L,M}R_v \subseteq^{\#} P_L).$$

Revealing the right hand side of the switchboard ensures that marks are associated to the correct candidate, i.e. the data on the switchboard matches the announced tally. This corresponds to the check

$$\psi_{\text{tally}} := t \in \bigcap_{P\in\mathcal{P}_{auth}} Ind_P^{qu}(\{\pi_S\sigma_{M=marked}\}).$$

The analysis of soundness and completeness of the verification procedure is hence analogous to Eperio, which implies that Scantegrity has the same weaknesses as Eperio. In particular, Scantegrity neither prevents ballot-stuffing nor does it ensure that each voter receives a distinct ballot serial number.

As in Eperio, Scantegrity's verification procedure becomes sound and complete with the additional checks ψ_{unique} and $\psi_{\text{no-stuffing}}$ from Section 4.4. Furthermore the privacy analysis conducted in Section 5.2 for Eperio is also valid for Scantegrity. Again $Choice = \{M, S\}$ and for each voter $v \in \mathcal{V}$, her vote is given by $\sigma_{V=v}\pi_{M,S}\overline{V}$. So the action of an element $\tau \in S_{\mathcal{V}}$ is given by swapping the tuples of sort $\{M, S\}$ of one voters' table with the tuples of sort $\{M, S\}$ of another voter's table. Analogously to what is shown in Section 5.2, the elements of sort $Choice = \{M, S\}$ of ballots in $\overline{C}$ are $G_{0,1}$-indistinguishable under the query set $Q = \{\pi_{ID,L,M}, \pi_{M,S}\}$ and constraints $\mathcal{P}_{con}$ for the same group $G_{0,1}$ that swaps elements that belong to marked and unmarked tuples respectively. This implies that Scantegrity's verification procedure guarantees vote privacy provided that all voters make the same number of marks.

This example illustrates the applicability of our approach to table-based voting protocols where the verification steps are not conducted directly on tables but instead using other means that can be modeled with tables.

7 RANDOM SAMPLE ELECTIONS

In Section E of the appendix, we present our analysis of the random sample election protocol proposed by David Chaum [9]. In a random sample election, only a small percentage of the electorate, chosen at random, is selected to vote. Chaum describes his protocol using a table format that can be seen as an extended variant of Eperio with a number of twists. In particular, there is a different ballot verification technique and so-called *decoy* votes are used to counter some forms of voter coercion. Decoy ballots are later ignored in the final tally, and can thus be freely given to attackers trying to buy votes. Unlike Eperio and Scantegrity, the verification procedure ensures that each voter receives a distinct ballot number and avoids ballot-stuffing.

As with the previous protocols, this case study illustrates how our methodology provides a systematic approach to analyzing table-based voting protocols. Moreover, its use also reveals weaknesses in the protocol. Namely, the protocol checks do not detect if decoy and non-decoy ballot are interchanged, which can be exploited by a dishonest authority to remove ballots from the tally.

8 RELATED WORK

8.1 Relational database theory

Our tables and operators differ in three ways from standard relational database theory [1]. First, our tables are multisets of tuples rather than sets of tuples. Second, our unknown value ($\perp$) differs from the standard NULL value. Third, our main object of study are table properties (sets of tables) rather than the individual tables.

The standard definition of NULL [25] stems from Codd's unknown value '@' [14] and his extension [15] of his relational data model [13]. This treatment of unknown values does not always lead to satisfactory results and many alternatives have been proposed. We refer to [33] for a survey. Our approach to unknown values differs from previous approaches in that it is tailored to a different and narrow purpose, i.e., the combination of data in order to detect conflicting information and measure the indistinguishability of possible combinations. Our merge operator $\textcircled{\wedge}$ is a variant of a full outer join database operator that (1) is defined on multisets of tuples, (2) enables comparing unknown and known values using the unknown value $\perp$, and (3) returns a set of all tables that combine the information present in the operands. In contrast, a full outer join operator is defined on relations and returns a single table.

8.2 Verifiability definitions

In our work, the global integrity property ϕ specifies the correctness of the election's outcome. Numerous other definitions have been proposed in the literature; see [17] for an overview. Our definition (Definition 4) compares to the guidelines from [17] as follows.

The qualitative goal from [17] essentially requires that ϕ holds for an execution if and only if there is a multiset of valid choices C such that the tallied result of C equals the announced result, and C (compared to the voter's intentions) consists of actual choices of honest voters that successfully performed their checks plus a subset of actual choices of honest voters that did not perform their check (successfully) and at most a fixed number of additional choices. Our definition of verifiability follows the same ideas, but differs in some details. First, we focus on tables, not on executions. Although this difference makes a precise comparison difficult, it is not a fundamental limitation of our approach as execution data can be cast into tables. Second, we explicitly account for inconsistent

views inside the tables, whereas [17] implicitly assumes that there is at least a consistent global view of the protocol execution (which however might include contradicting events emitted by different or dishonest parties). Finally, our definition does not account for dishonest voters, as we do not know a priori which data in the tables originates from honest participants, like in the real world. Handling this would require a dispute resolution procedure deciding which data can or must be ignored during the verification process, which is outside the scope of our work.

8.3 Privacy definitions

Measuring privacy when information in tables is revealed is a general problem, relevant in many domains. For example, for data sets from clinical studies, genomics, and social networks, one would like to release person-specific data for research, without compromising the data subjects' privacy. Researchers have observed that removing personally identifying information from such data sets is insufficient as relinkage is possible using publicly available outside information [23, 28, 29].

In these domains, the privacy problem differs from that in voting. In such data sets, all entries are revealed and personal identifying information is removed. In contrast, table-based voting protocols typically deal with tables containing personally identifying information (e.g., voters' identities), where not all entries are simultaneously revealed. In the former case, the privacy issue is to relink the table entries to specific individuals using *external* information, in the latter we are concerned with linking the personally identifying information present in the table to the other entries *inside* the table. Hence a notion like G-indistinguishability is well suited for voting protocols as it analyzes linkages inside the table.

One can map other privacy notions onto G-indistinguishability by adding extra columns to the original table to represent the external information within the extended dataset. Consider, for example, k-anonymity [32], which measures privacy by a parameter k, and states that any quasi-identifier (a collection of attributes that can serve to identify individuals from external information) must be present in at least k rows. Given a table T, if we add an extra column with uniquely identifying keys (representing the external information that could uniquely identify each row), k-anonymity for the table T constitutes a special case of G-indistinguishability under the query revealing the quasi-identifiers and the added column with the uniquely identifying keys, for a group G that is the product of independent symmetric groups, each of size at least k.

Note that much of the thrust in privacy metrics and in other approaches such as l-diversity [27], t-closeness, and differential privacy [20], is on the mechanism side: how to transform data by generalization or transformations like adding noise. Such privacy notions and mechanisms are inappropriate in a voting context where, in particular, the modification of sensitive data would change the election results.

9 CONCLUSION

Table-based procedures, with their simple checks, hold the promise of increasing the electorate's trust and acceptance of e-voting. However, despite this apparent simplicity, an analysis methodology has been lacking, with the consequence that existing proposals suffer from weaknesses and missing assumptions. Our results fill this gap.

As followup work, we intend to build proof support for our methodology within a higher-order logic theorem prover. This would entail defining our operators and deriving their properties. The result would enable machine-supported correctness proofs, following the steps laid out in our methodology. Interesting too, would be to use our methodology to support the *design* of new table-based protocols, e.g., exploring different ways of decomposing a global integrity property ϕ into individual checks. Finally, extending our privacy definitions to receipt-freeness is another practically relevant direction for future work.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu (Eds.). 1995. *Foundations of Databases: The Logical Level* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] David A. Basin, Sasa Radomirovic, and Lara Schmid. 2020. Dispute Resolution in Voting. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*. IEEE, 1–16.

[3] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *Advances in Cryptology – EUROCRYPT 2012*, David Pointcheval and Thomas Johansson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 263–280.

[4] Susan Bell, Josh Benaloh, Michael D. Byrne, Dana Debeauvoir, Bryce Eakin, Philip Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. 2013. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13)*. USENIX Association, Washington, D.C. https://www.usenix.org/conference/evtwote13/workshop-program/presentation/bell

[5] Josh Daniel Cohen Benaloh. 1987. *Verifiable Secret-Ballot Elections*. Ph.D. Dissertation. Yale University, USA. AAI8809191.

[6] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 499–516. https://doi.org/10.1109/SP.2015.37

[7] C. Burton, C. Culnane, and S. Schneider. 2016. vVote: Verifiable Electronic Voting in Practice. *IEEE Security & Privacy* 14, 04 (jul 2016), 64–73. https://doi.org/10.1109/MSP.2016.69

[8] Swiss Federal Chancellery. [n.d.]. Anforderungskatalog für eidgenössische Volksabstimmungen mit der elektronischen Stimmabgabe. https://www.bk.admin.ch/themen/pore/evoting/07979/index.html

[9] David Chaum. [n.d.]. Random-Sample Voting - More democratic, better quality, and far lower cost. https://rsvoting.org/whitepaper/white_paper.pdf. Last accessed on 2021-01-20.

[10] David Chaum. 1988. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA. In *Advances in Cryptology — EUROCRYPT '88*, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and Christoph G. Günther (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–182.

[11] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. 2008. Scantegrity II: End-to-end Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes. In *Proceedings of the Conference on Electronic Voting Technology* (San Jose, CA) *(EVT'08)*. USENIX Association, Berkeley, CA, USA, Article 14, 13 pages. http://dl.acm.org/citation.cfm?id=1496739.1496753

[12] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and P. Vora. 2008. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security Privacy* 6, Article 3, 40–46 pages.

[13] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387. https://doi.org/10.1145/362384.362685

[14] E. F. Codd. 1975. Understanding Relations (Installment #7). *FDT Bull. ACM SIGFIDET SIGMOD* 7, 3 (1975), 23–28.

[15] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions of Database Systems* 4, 4 (1979), 397 – 434.

[16] Josh D. Cohen and Michael J. Fischer. 1985. A Robust and Verifiable Cryptographically Secure Election Scheme. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (SFCS '85)*. IEEE Computer Society, USA, 372–382. https://doi.org/10.1109/SFCS.1985.2

[17] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. 2016. SoK: Verifiability Notions for E-Voting Protocols. In *IEEE*

Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. IEEE Computer Society, 779–798. https://doi.org/10.1109/SP.2016.52

[18] German Federal Constitutional Court. 2009. Use of voting computers in 2005 Bundestag election unconstitutional. Press Release No. 19/2009, available at https://www.bundesverfassungsgericht.de/SharedDocs/Pressemitteilungen/EN/2009/bvg09-019.html. Last visited 2021-01-18.

[19] Stéphanie Delaune, Steve Kremer, and Mark Ryan. 2010. *Verifying Privacy-Type Properties of Electronic Voting Protocols: A Taster*. Springer Berlin Heidelberg, Berlin, Heidelberg, 289–309. https://doi.org/10.1007/978-3-642-12980-3_18

[20] Cynthia Dwork. 2006. Differential Privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006) (Lecture Notes in Computer Science, Vol. 4052)*. Springer Verlag, Venice, Italy, 1–12. http://research.microsoft.com/apps/pubs/default.aspx?id=64346

[21] Aleksander Essex, Jeremy Clark, and Carlisle Adams. 2010. Towards Trustworthy Elections, New Directions in Electronic Voting. In *Towards Trustworthy Elections, New Directions in Electronic Voting (Lecture Notes in Computer Science, Vol. 6000)*. Springer, 388–401.

[22] Aleksander Essex, Jeremy Clark, Urs Hengartner, and Carlisle Adams. 2010. Eperio: Mitigating Technical Complexity in Cryptographic Election Verification. In *Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (Washington, DC) *(EVT/WOTE'10)*. USENIX Association, Berkeley, CA, USA, 1–16. http://dl.acm.org/citation.cfm?id=1924892.1924905

[23] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. 2008. Resisting Structural Re-Identification in Anonymized Social Networks. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 102–114. https://doi.org/10.14778/1453856.1453873

[24] Martin Hirt and Kazue Sako. 2000. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *Advances in Cryptology — EUROCRYPT 2000*, Bart Preneel (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 539–556.

[25] ISO/IEC 9075:2016 2016. *Information technology — Database languages — SQL*. Standard. International Organization for Standardization, Geneva, CH.

[26] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov (Eds.). ACM, 526–535. https://doi.org/10.1145/1866307.1866366

[27] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. L-diversity: Privacy Beyond K-anonymity. *ACM Trans. Knowl. Discov. Data* 1, 1, Article 3 (March 2007).

[28] Bradley Malin and Latanya Sweeney. 2004. How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems. *Journal of Biomedical Informatics* 37, 3 (2004), 179 – 192. https://doi.org/10.1016/j.jbi.2004.04.005

[29] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust De-Anonymization of Large Sparse Datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (SP '08)*. IEEE Computer Society, USA, 111–125. https://doi.org/10.1109/SP.2008.33

[30] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and Its Application to E-Voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security* (Philadelphia, PA, USA) *(CCS '01)*. Association for Computing Machinery, New York, NY, USA, 116–125. https://doi.org/10.1145/501983.502000

[31] Stefan Popoveniuc and Benjamin Hosp. 2010. An Introduction to PunchScan. In *Towards Trustworthy Elections, New Directions in Electronic Voting (Lecture Notes in Computer Science, Vol. 6000)*. Springer, 242–259.

[32] Latanya Sweeney. 2002. k-ANONYMITY: A MODEL FOR PROTECTING PRIVACY. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

[33] Ron van der Meyden. 1998. *Logical Approaches to Incomplete Information: A survey*. Springer Berlin Heidelberg, 307–356.

A PRELIMINARY LEMMAS

The following lemma summarizes properties of $[T]$ and $[T]^N$, which we will use later.

Lemma A.1. Let $S, T \in \mathcal{T}_\perp$ be tables and N an arbitrary sort. Then

(1) If $T \in \mathcal{T}$ then $T \in [T]$.
(2) If $T \in [S]$ then $[T] \subseteq [S]$.
(3) $[T]^N \subseteq [T]$
(4) If $[T] \subseteq [S]$, then $[T]^N \subseteq [S]^N$.
(5) If $T \in \mathcal{T}$ and $S \subseteq^\#_\perp T$ then $[T] \subseteq [S]$.
(6) If $\forall T' \in \mathcal{T} : T' \doteq T \Rightarrow S \subseteq^\#_\perp T'$ then $[T] \subseteq [S]$.
(7) If $S = \pi_N(T)$ then $[T] \subseteq [S]$.

Proof. Properties (1) – (4) and (7) follow immediately from their definitions. It remains to show (5) and (6).

(5) Let $S' \in \mathcal{T}$ such that $S' \doteq S$ and $S' \subseteq^\# T$. Then $T \in [S'] \subseteq [S]$, thus $[T] \subseteq [S]$ by property (2).
(6) Let $T'' \in [T]$. By definition, $T \subseteq^\#_\perp \pi_{\mathrm{sort}(T)}(T'')$. Thus, there is $T' \in \mathcal{T}$ such that $T' \doteq T$ and $T' \subseteq^\# \pi_{\mathrm{sort}(T)}(T'')$. Therefore $T'' \in [T']$. It follows that

$$[T] \subseteq \bigcup_{\substack{T' \in \mathcal{T} \\ T' \doteq T}} [T'].$$

Suppose that $\forall T' \in \mathcal{T} : T' \doteq T \Rightarrow S \subseteq^\#_\perp T'$. Then

$$[T] \subseteq \bigcup_{\substack{T' \in \mathcal{T} \\ T' \doteq T}} [T'] \subseteq [S],$$

where the second inclusion follows from Property (5). Thus $[T] \subseteq [S]$. □

The following lemma follows immediately from Definition 1.

Lemma A.2. Let Q_1, Q_2 be sets of queries and B a set of properties. Then

$$Ind_T(Q_1 \cup Q_2, B) = Ind_T(Q_1, B) \cap Ind_T(Q_2, B).$$

B PROOFS FOR SECTION 3

Lemma 1. Let $T_1, \ldots, T_k \in \mathcal{T}$, and $Key \neq \emptyset$ be a set of column names such that *Key* is a subsort of each of these tables. Then there exists a table $Z \in \mathcal{T}_\perp$ such that $\bigwedge^{Key}\{T_1, \ldots, T_k\} = [Z]^{Key}$.

Proof. We give an explicit construction of such a table Z. For readability of formulae let $N = Key$. If $k = 1$, the statement is trivial as by definition $\bigwedge^N\{T\} = [T]^N$. We may hence assume $k > 1$. Observe that $\owedge^N$ is an associative and commutative operator. Therefore it suffices to prove the hypothesis for two known tables T_1 and T_2. The lemma then follows by induction.

We consider two cases. The first case is that $T_1 \owedge^N T_2 = \emptyset$. Let $Z = Z_0 \in \mathcal{T}$ be a table of sort N that contains a tuple of multiplicity 2. Then $[Z_0]^N = \emptyset$.

In the second case, $T_1 \owedge^N T_2 \neq \emptyset$ $(*)$. We construct a table Z such that $[Z]^N = T_1 \owedge^N T_2$, by matching common subtuples of T_1 and T_2, then horizontally extending T_1 and T_2 to include each other's missing columns, and filling up missing entries with the unknown value $\perp$.

Let $M = \mathrm{sort}(T_1) \cap \mathrm{sort}(T_2)$ and note that $N \subseteq M$, since $N \subseteq \mathrm{sort}(T_1)$ and $N \subseteq \mathrm{sort}(T_2)$ by hypothesis. By $(*)$ the multiplicity of each tuple in $\pi_M(T_1)$ and $\pi_M(T_2)$ is 1. Thus, for $Z_M = \mathrm{set}(\pi_M(T_1)) \cup \mathrm{set}(\pi_M(T_2))$ we have $\pi_N(Z_M) = \mathrm{set}(\pi_N(Z_M))$. Hence, for each $i \in \{1, 2\}$, the function $f_i : \mathrm{set}(T_i) \to \mathrm{set}(Z_M)$, such that $f_i(z) = \pi_M(z)$, is an injective function. Let g_1 be a function from $\mathrm{set}(Z_M)$ to the set of tuples of the sort $\mathrm{sort}(T_1) \setminus \mathrm{sort}(T_2)$ defined by $g_1(z) = \pi_{\mathrm{sort}(T_1)\setminus\mathrm{sort}(T_2)} f_1^{-1}(z)$ if the pre-image exists, and the tuple of unknown values otherwise. We define $g_2(z)$ analogously. Let $Z = \{g_1(z) \cdot z \cdot g_2(z) \mid z \in Z_M\}$, where $\cdot$ denotes tuple concatenation.

It remains to show that $[T_1]^N \cap [T_2]^N = [Z]^N$. Let $i \in \{1, 2\}$. By the construction of Z, $\forall Z' \in \mathcal{T} : Z' \doteq \pi_{\mathrm{sort}(T_i)}(Z) \Rightarrow T_i \subseteq^\# Z'$. By Lemma A.1 (6) and (7), $[Z] \subseteq [\pi_{\mathrm{sort}(T_i)}(Z)] \subseteq [T_i]$. Thus $[Z] \subseteq [T_1] \cap [T_2]$. By Lemma A.1 (4), $[Z]^N \subseteq [T_1]^N \cap [T_2]^N$. Conversely, let $T \in [T_1]^N \cap [T_2]^N$. Then $Z_M \subseteq^\# \pi_M(T)$. Let T_M be the table of tuples in T that horizontally extend a tuple in Z_M. Since each tuple of sort N is unique in T, it follows that $\pi_M(Z) = Z_M = \pi_M(T_M)$. By the construction of Z, $Z \doteq \pi_{\mathrm{sort}(Z)}(T)$. Thus $T \in [Z]$. □

C PROOFS FOR SECTION 4

Lemma 2. Let $\mathcal{R} \subseteq \mathcal{T}$ be a set of tables, defining a property, and $\{S_0, S_1, S_2, ..., S_k\}$ a subsort of $\mathrm{sort}(R)$ for all $R \in \mathcal{R}$. Let $T \in \mathcal{T}$ be a table such that $T \in \owedge^{\{S_0\}}\{\pi_{S_0,S_i} R \mid R \in \mathcal{R}\}$ for all $i \in \{1, \ldots, k\}$. Then

$$T \in \owedge^{\{S_0\}}\{\pi_{S_0,S_1,..,S_k} R \mid R \in \mathcal{R}\}.$$

Proof. Let $t = \langle S_0 : s_0, \quad S_1 : s_1, \ ..., \quad S_k : s_k \rangle$ be a tuple in $\pi_{S_0,S_1,..,S_k} R$ for some table $R \in \mathcal{R}$. We need to show it is also a tuple of T. By hypothesis, for each i, the subtuple $t_i = \langle S_0 : s_0, S_i : s_i \rangle$ is a tuple of $\pi_{S_0,S_i} T$. However, since there is at most one tuple t' in T with $t'(S_0) = s_0$, then $t_i = \pi_{S_0,S_i} t'$ for each i, so $t' = t$ is indeed a tuple of T. □

Theorem C.1. The Eperio verification procedure under the additional assumptions of Section 4.4.2 is complete: If ψ_i does not hold, for some $i \in \{0, 1, \mathrm{tally}, \text{print-audit}, \mathrm{unique}, \text{no-stuffing}\}$, then neither does ϕ.

Proof. (1) The check ψ_0 states that

$$\owedge^{\{U\}} \mathcal{P}_{auth} \cap \bigcap_{P \in \mathcal{P}_{auth}} shape_P \neq \emptyset.$$

Since C is a subset of this set, $\neg\psi_0$ implies $C = \emptyset$.

(2) If the check ψ_1 does not hold, then there is a voter $w \in \mathcal{V}$ and a table $P \in \mathcal{P}_{auth}$ with $\pi_{U,M} P \in \mathcal{T}$ such that

$$\pi_{U,M} R_w \not\subseteq^\# \pi_{U,M} P.$$

Let $\langle U : u_w, M : m_w \rangle \in R_w$ be the tuple that is not in P. Suppose u_w does not appear in $\pi_{U,M} P$, then any table in $\owedge^{\{U\}}(\mathcal{P}_{auth} \cup Views)$ must contain all the unique serial numbers in P and an extra row containing u_w. Any such table does not have the shape of P, so $\mathcal{P}_{con}$ would not be satisfied and C would be empty.

Now suppose u_w appears in $\pi_{U,M} P$ but m_w does not. Then it is not possible to merge tuple $\langle U : u_w, M : m_w \rangle \in R_w$ with a different tuple $\langle U : u_w, M : m_p \rangle \in \pi_{U,M} P$ while maintaining unique U values; hence C would again be empty.

(3) If ψ_{tally} does not hold, then for some $P, t \notin Ind_P^{qu}(\{\pi_S\sigma_{M=1}\})$. Clearly any table that extends such $P \in \mathcal{P}_{auth}$ does not belong to $Ind_t^{qu}(\{\pi_S\sigma_{M=1}\})$, hence not to $\mathcal{P}_{con}$, and again $C = \emptyset$.

(4) By definition $\neg\psi_{\text{print-audit}}$ states that

$$\bigwedge^{\{U\}}(\mathcal{P}_{auth} \cup \{\pi_{U,S}R \mid R \in Views\}) \bigcap_{P\in\mathcal{P}_{auth}} shape_P = \emptyset.$$

In this case C is empty too as it is a subset of the empty set described above.

(5) Suppose ψ_{unique} does not hold. Then $\bigwedge^{Key}\{\pi_{V,U}R \mid R \in Views\} = \emptyset$ and hence clearly also $\bigwedge^{Key} Views$ is empty. Again this implies C is empty as it is a subset of the former.

(6) Suppose the $\psi_{\text{no-stuffing}}$ is not satisfied. Then the number of marks in $\overline{V}$ is different than that of one of the tables in $\mathcal{P}_{auth}$, say P. If C is not empty, this implies that for any table $T \in C$, T must have the same number of marks as P (since T both extends P and is of its same shape). Hence $\pi_S\sigma_{M=1}T \neq \pi_S\sigma_{M=1}\overline{V}$ simply because they are of different shape, so the second predicate of ϕ does not hold. Otherwise C is empty and ϕ does not hold either.

□

D PROOFS FOR SECTION 5

Theorem 1. Suppose that $\mathcal{P}_{auth} = \{q(\mathcal{E}) \mid q \in Q\}$ and $(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con}) \neq \emptyset$. Then a protocol's verification procedure guarantees vote privacy as in Definition 5 provided that for any $\bar{C} \in \min(\bigwedge^{Key}(\mathcal{E} \cup Views) \cap \mathcal{P}_{con})$ the tuples of sort *Choice* of $\bar{C}$ are $S_{\mathcal{V}}$-indistinguishable under the queries in Q and that $\tau(\bar{C}) \in \mathcal{P}_{con}$ for all $\tau \in S_{\mathcal{V}}$.

Proof. Fix a $\bar{C} \in \min \bigwedge^{Key}(\mathcal{E} \cup Views)$ and a $\tau \in S_{\mathcal{V}}$. By definition of τ's operation on *Views*, $\tau(\bar{C}) \in [\tau(R_v)]$ for all $R_v \in Views$. By definition of the merge operator, $\bar{C} \in [\mathcal{E}]$ and hence $\bar{C} \in [q(\mathcal{E})]$ for all $q \in Q$. By definition of $S_{\mathcal{V}}$-indistinguishability, for all $q \in Q$: $q(\tau(\bar{C})) = q(\bar{C})$. We thus obtain that $\tau(\bar{C}) \in [q(\mathcal{E})]$ for all $q \in Q$. Using that $\mathcal{P}_{auth} = \{q(\mathcal{E}) \mid q \in Q\}$ gives

$$\tau(\bar{C}) \in \bigwedge^{Key}(\mathcal{P}_{auth} \cup \tau(Views)).$$

Since by hypothesis all permutations of votes satisfy the constraints in $\mathcal{P}_{con}$, we also have $\tau(\bar{C}) \in \mathcal{P}_{con}$ and therefore

$$\bigwedge^{Key}(\mathcal{P}_{auth} \cup \tau(Views)) \cap \mathcal{P}_{con} \neq \emptyset.$$

□

Lemma 3. In Eperio, the elements of sort *Choice* $= \{M, S\}$ in $\bar{C}$ are $G_{0,1}$-indistinguishable under the query set $Q = \{\pi_U, \pi_{U,M}, \sigma_{M=-1}, \pi_{M,S}\}$, where $G_{0,1}$ is the product of two groups that permute tuples of sort $\{M, S\}$. The first group arbitrarily permutes elements of sort $\{M, S\}$ that belong to a tuple whose value of sort M is 0, the second group those whose value of sort M is 1.

Proof. To compute the group under which *Choice* $= \{M, S\}$ elements are indistinguishable, we must analyze the indistinguishability set $Ind_{\bar{C}}(Q, \{shape_{\bar{C}}\})$. We can achieve this by computing the sets $H_i = Ind_{\bar{C}}(\{q_i\}, \{shape_{\bar{C}}\})$ for $q_i \in Q$ and the groups G_i under which the elements of columns $\{M, S\}$ are indistinguishable in H_i. We then apply Lemma D.1.

(1) $H_0 = Ind_{\bar{C}}(\{\pi_U\}, \{shape_{\bar{C}}\})$ contains all tables with the same number of rows as $\bar{C}$ in which all $t \in \pi_U(\bar{C})$ occur exactly once. Hence $G_0 = P_{\bar{C}}(\{M, S\})$, since the columns M and S are not observed.

(2) $H_1 = Ind_{\bar{C}}(\{\pi_{U,M}\}, \{shape_{\bar{C}}\})$ contains all tables with tuples of sort $\{U, M\}$ that are equal to those of $\bar{C}$. Since U values are unique, this consists of all tables where the M value associated to each U value does not change. G_1 is hence the product of three subgroups J_1, J_0, and J_{-1} of $P_{\bar{C}}(\{M, S\})$ that permute the tuples of sort $\{M, S\}$ that contain 1, 0, and -1, respectively, in column M.

(3) $H_2 = Ind_{\bar{C}}(\{\sigma_{M=-1}(\bar{C})\}, \{shape_{\bar{C}}\})$ contains all tables that are the disjoint union of the multiset $\sigma_{M=-1}(\bar{C})$ and a multiset of tuples of sort sort$(\bar{C})$ whose subtuples of sort $\{M\}$ do not contain -1. Hence G_2 is the permutation group of elements in columns M and S that are not part of a print-audited ballot.

(4) $H_3 = Ind_{\bar{C}}(\{\pi_{M,S}\}, \{shape_{\bar{C}}\})$ contains all tables that contain the same columns of sort $\{M, S\}$ as $\bar{C}$. Since tables are row-wise unordered, $G_3 = P_{\bar{C}}(\{M, S\})$.

By Lemma D.1, the elements in column S are $G_{0,1}$-indistinguishable, where $G_{0,1}$ is the product of the groups J_1 and J_0. This is because $G_{0,1}$ is a subgroup of each of the groups G_0, G_1, G_2 and G_3. □

Lemma D.1. Let G_1 and G_2 be subgroups of $P_T(N)$ for $N \subseteq \text{sort}(T)$. If the entries of a table T are G_1-indistinguishable under the queries Q_1 and G_2-indistinguishable under the queries Q_2, then the entries are $G_1 \cap G_2$-indistinguishable under the queries $Q_1 \cup Q_2$.

Proof. Let $g \in G_1 \cap G_2$. Then $g(T) \in Ind_T(Q_1, \{shape_T\})$ and $g(T) \in Ind_T(Q_2, \{shape_T\})$. Thus by Lemma A.2, $g(T) \in Ind_T(Q_1 \cup Q_2, \{shape_T\})$. □

E RANDOM SAMPLE ELECTIONS

David Chaum proposed random sample elections [9] as a way to run elections where only a small percentage of the electorate, chosen at random, are selected to vote. The list of all registered voters is public and is referred to by Chaum as the *voter roster*. He formulated the protocol in a table format, and it can be seen as an extended variant of Eperio that supports decoy votes to counter some forms of coercion. As with Eperio, this case study illustrates how our methodology provides a systematic approach to analyzing table-based voting protocols.

E.1 Election procedure

The election and verification procedure revolves around a series of encrypted permutations of a $2u \times 8$ table $\mathcal{E}$, where u denotes the pre-determined total number of ballots. To prevent vote buying, the protocol dictates that a fixed number of these ballots, called *decoy* ballots, are ignored in the final tally, and can thus freely be given to attackers trying to buy votes. Voters and auditors should not be able to prove which ballots are decoys. Randomly chosen voters receive two (non-decoy) ballots, which should each be labeled with unique serial numbers, one used for voting, the other for print auditing. Voters may also ask for a decoy ballot. Each column of each permutation of the table is encrypted with a different unique

secret key. The authority later reveals some of these keys for some subset of permutations for the audit, as in Eperio.

The election process consists of nine steps, decomposed into four phases. The steps are illustrated in Figure 6.

Phase 1 *Prior to the election (steps 1 - 3 in Figure 6):*
The encryption (each column with a distinct unique secret key) of $5x$ row-permutations of columns 1, 3, 5, and 6 of the table $\mathcal{E}$, are generated and published (Step 1), where x is a fixed parameter. Columns 1 and 3 indicate ballot serial numbers and their respective candidate/vote (for example YES or NO). Columns 5 and 6 indicate which of these ballots are decoy and the rows corresponding to real ballots contain random numbers whose sum is used to determine the voter to whom the ballot will be sent. The sum of these numbers must be the same for each copy of $\mathcal{E}$, but the summands themselves differ for each different permutation.

A public random draw, i.e., a draw from a public source of randomness, is then executed (Step 2) and the result is published (unencrypted) as table $\mathbb{D}$, which is a $u \times 2$ table that links ballot numbers to a random summand used to select voters. The sum of this number and the elements of columns 5 and 6 corresponding to the same ballot row, determine the position of a voter in the voter roster. The fact that the random draw is public and that commitments of columns 5 and 6 are published before the draw (hence cannot be modified) ensures that the selected sample of voters is truly random.

Columns 7 and 8 are then completed by the authority by copying the values of columns 5 and 6 in the case of non-decoy votes, and in the case of decoy votes by computing the number such that its sum with the corresponding row in $\mathbb{D}$ corresponds to the position of a voter who requested the decoy vote. Finally encryptions of the $5x$ row-permutations (using the same permutations as for the previous columns) of these columns are also published, and the ballots are printed and mailed to the voters (Step 3).

Phase 2 *During the election (Step 4):*
Each voter in the sample receives two ballots, each identified by a unique serial number. He chooses one of them to vote, and audits the second one (Step 4 in Figure 6). Ballots associate vote options to vote codes. To vote, each voter publishes the ballot's serial number together with the vote code on a public platform (called the bulletin board *BB*). To audit a ballot, he publishes the serial number of the ballot together with both vote codes and their associated vote on the bulletin board.

Phase 3 *After the election, prior to the audit (Step 5):*
Encryptions of columns 2 and 4 and their $5x$ permutations are generated from the bulletin board and published (Step 5). These columns indicate which votes are print audited and for the non-audited ones, they indicate the voters' choices.

Phase 4 *Audit (Steps 6 - 9):*
The auditors perform a cut & choose game over the $5x$ permutations of the table $\mathcal{E}$ published in Phases 1 and 3, also using the bulletin board and the random draw. The $5x$ encrypted tables are randomly split into five batches (each of x tables) using the public random draw that is executed in Step 6. As indicated in the bottom of Figure 6, for all tables in the first batch the columns 1, 2, and 4 are revealed, for the second batch columns 2 and 3 are revealed, for the third batch columns 3, 4, 5, and 7, for the fourth batch columns 3, 4, 6, and 8, and finally for the fifth batch columns 1, 7, and 8. We will detail the verification steps (Steps 7 to 9) in the following section.

In his paper, Chaum informally describes the properties his protocol achieves: votes are counted as intended and kept private. However, no proofs are given. We illustrate next how our methodology again leads to clear and rigorous analysis of correctness and privacy.

E.2 Model and verification procedure

As with Eperio, we analyze the soundness of the verification of the integrity property ϕ. However, we abstract slightly the handling of the summands in the last columns of the table. Namely, we model Chaum's 8 column table using a 6 column table $\mathcal{E}$ of sort $\{\mathbb{1}, \mathbb{2}, \mathbb{3}, \mathbb{4}, \mathbb{5}, V\}$:

- The first four columns match Chaum's table columns.
- Columns 5 and 6 of Chaum's table are only used in the protocol to distinguish decoy votes from non-decoy ones and to ensure the randomness of the voter selection. To maintain the anonymity of non-decoy ballots, these two columns are never revealed at the same time. Since the random summands for non-decoy ballots can be any number and can be different in different copies of the table, the only information obtained by revealing only one of the columns 5 or 6 is whether ballots are decoy or not. In our model, we model this by merging columns 5 and 6 in a unique column of sort $\mathbb{5}$, whose elements are either the value *"decoy"* as indicated in the protocol or the value *"non-decoy"*, instead of a random summand.
- For the same reasons as for columns 5 and 6, and since the sums of columns 7 and 8 are not directly linked to decoy or non-decoy ballots, we merge columns 7 and 8, to a unique column of sort V containing the numbers of voters as indicated by $\mathbb{D}$. This still allows auditors to check that ballots have been sent to the voter chosen in the public draw $\mathbb{D}$.

Since the values in columns 5 and 6 can be chosen arbitrarily by the election authority, the only source of randomness for the selection of voters comes from the public draw $\mathbb{D}$. In our modeling of $\mathbb{D}$, we will again abstract away the summands and model the table $\mathbb{D} \in \mathcal{T}$ as a $2u \times 2$ table of sort $\{\mathbb{1}, V\}$ where for each row corresponding to a ballot serial number, V directly contains the number of the voter linked to the ballot (instead of a random summand). We also assume that the elements in V corresponding to a real ballot are indeed random, but note that this is only important for the randomness of the sample, not for the correctness of the result.
Finally we represent the bulletin board $BB \in \mathcal{T}_\perp$ as a $2u \times 4$ table of sort $\{\mathbb{1}, \mathbb{2}, \mathbb{3}, \mathbb{4}\}$, where we have unknown values in column 3 for votes that are not print audited.
The permutations of the table $\mathcal{E}$ are grouped in 5 batches of x tables each, and we denote each batch by $P^{(k)} = \{P_1^{(k)}, ..., P_x^{(k)}\} \in \mathcal{T}_\perp$ for $1 \leq k \leq 5$. Each batch has unknown values for the encrypted values, and known values in the columns whose secret key has been revealed, as described in Phase 4. In particular, the fifth batch will have known values for columns $\mathbb{1}$ and V. As the table $\mathbb{D}$ is public and unencrypted, revealing column V in the batch where in Chaum's table both column 7 and 8 are revealed, correctly represents Chaum's protocol.
We denote the union of all these batches and the publicly available

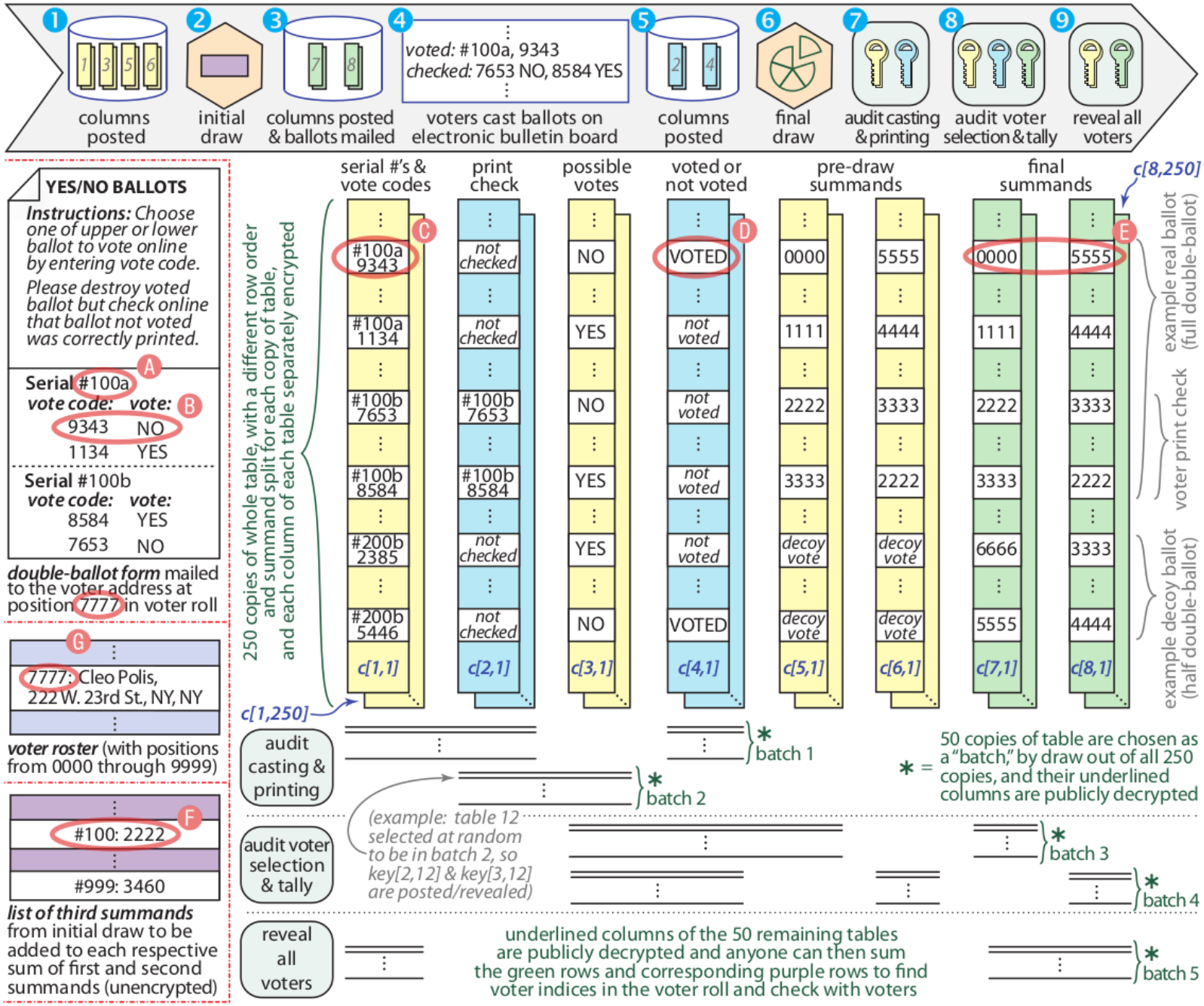

Figure 6: Random Sample Elections, from [9], reproduced with permission from the author.

information by

$$\mathcal{P}_{batches} = \{P_1^{(1)}, ..., P_x^{(1)}, P_1^{(2)}, ..., P_x^{(2)}, ..., P_x^{(5)}\},$$
$$\mathcal{P}_{auth} = \{\mathbb{D}, BB\} \cup \mathcal{P}_{batches}.$$

The outcome function is given by $tal := \pi_{\mathbb{3}}\sigma_{\mathbb{4}=voted}\sigma_{\mathbb{5}=non\text{-}decoy}$ and may hence be computed from the third or fourth batch. Let $shape_{\mathcal{P}} = \bigcap_{P\in\mathcal{P}_{batches}}\{shape_P\}$. If t indicates the outputted tally, we have

$$\mathcal{P}_{con} = Ind_t^{qu}(\{tal\}) \cap shape_{\mathcal{P}}.$$

Let $\mathcal{V}$ be the random sample of voters who receive a ballot (decoy or not). Each voter's view is represented by a table $R_v \in \mathcal{T}$ that contains information about their double ballot and hence is of sort $\{V, \mathbb{1}, \mathbb{2}, \mathbb{3}, \mathbb{4}, \mathbb{5}\}$, where V again indicates the voter's position in the voter roster and columns of sort $\mathbb{5}$ have a *"decoy"* value in the row of decoy ballots in case they have asked for a decoy ballot and the *"non-decoy"* value in case they believe their ballot is real. Let $Views = \{R_v \mid v \in \mathcal{V}\}$. As in Eperio, Lemma 1 implies that when the combination of these tables is consistent, then it is represented by a table $\overline{V} \in \mathcal{T}$ given by the unique element in the set $\min \bigwedge^{Key} Views$.

The unique key is the ballot number, of sort $\mathbb{1}$. Our goal is again to ensure that the election is counted as intended, as indicated by the property ϕ specified in Definition 4. As in Eperio, the protocol itself ensures that the shape of all tables is correct and that all serial numbers are unique. We again set $\mathcal{P} := \bigwedge^{\{\mathbb{1}\}}\mathcal{P}_{batches} \cap shape_{\mathcal{P}}$, and denote this verification step by $\psi_0 := \mathcal{P} \neq \emptyset$.

The verification steps detailed by Chaum are grouped in 3 sections (corresponding respectively to Steps 7, 8, and 9) as follows.

Audit casting and printing. Using tables from the first batch, everyone checks that the values in the columns 1, 2, and 4 of the tables in $P^{(1)}$ match the values published on the bulletin board BB. Since all tables in $\mathcal{P}$ extend the tables $\{\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}P_i^{(1)} \mid i = 1, ..., x\}$ and have the same shape, we conclude that $\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}BB = \pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}P$ for each table $P \in \mathcal{P}$. In his paper, Chaum assumes that the bulletin board correctly reflects the voter's views. In particular this means that for each voter $v \in \mathcal{V}$, $\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}R_v \subseteq^{\#} \pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}BB$. Combining with the above we may write this verification step as

$$\psi_1 := \bigwedge_{v\in\mathcal{V}}\bigwedge_{P\in\mathcal{P}} \pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}R_v \subseteq^{\#} \pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}P.$$

Using tables of batch 2, everyone can check that the second and third column of $\mathcal{E}$ match the audited ballots on the bulletin board. Since the choice of which ballots are print-audited is made after the publication of columns 1 and 3, we will make the assumption that this implies that the possible votes and vote codes on the ballots match the ones on the permuted tables for *all* ballots of column 1 and not only for the print audited ones of column 2. Again assuming correctness of BB, we may rewrite this verification step as

$$\psi_2 := \bigwedge^{\{\mathbb{1}\}}(\{\pi_{\{\mathbb{1},\mathbb{3}\}}BB\} \cup \{\pi_{\{\mathbb{1},\mathbb{3}\}}R \mid R \in \mathcal{R}\} \cup \mathcal{P}_{batches}) \cap shape_{\mathcal{P}} \neq \emptyset.$$

Audit voter selection and tally. In Step 8, batches 3 and 4 are used. From these checks, the announced tally t is verified (i.e. decoy votes are revealed) and voters verify that their choice has been counted. Thus we define ψ_t and ψ_3 to be the following properties

$$\psi_t := \quad t \in \bigcap_{P\in\mathcal{P}} Ind_P^{qu}(\{\pi_{\mathbb{3}}\sigma_{\mathbb{4}=voted}\sigma_{\mathbb{5}=non\text{-}decoy}\});$$
$$\psi_3 := \quad \bigwedge_{v\in V}\bigwedge_{P\in\mathcal{P}} (\pi_{\{\mathbb{3},\mathbb{4},\mathbb{5}\}}R_v \subseteq \pi_{\{\mathbb{3},\mathbb{4},\mathbb{5}\}}P).$$

As specified by Chaum, these batches should also be used to check that values in columns 7 and 8 correspond to the values in columns 5 and 6 and hence to verify the correctness of voter selection, but we have abstracted these verification steps away by merging columns 7 and 8 in the unique column V.

Audit all voters. Finally in step 9, batch 5 is used. Columns 1, 7 and 8 of Chaum's table (hence columns $\mathbb{1}$ and V in our model) reveal which voter corresponds to each ballot. Chaum specifies that these voters must be contacted to verify they have received the correct ballot and have effectively voted. This also ensures that no other votes have been added to the table. We model this check by

$$\psi_{\text{voter}} := \bigwedge^{\{\mathbb{1}\}}(\mathcal{P} \cup \{\pi_{\{V,\mathbb{1}\}}R \mid R \in Views\} \cup \mathbb{D}) \cap shape_{\mathcal{P}} \neq \emptyset \wedge \pi_{\{V,\mathbb{1}\}}\overline{V} = \mathbb{D}.$$

E.3 Soundness

As mentioned, to achieve soundness and completeness, we must assume correctness of the table $\mathbb{D}$ and of the bulletin board BB. For example, the table BB could contain an altered vote of an honest voter, resulting in the verification succeeding although the vote is incorrect. Similarly a corrupted voter could post a different vote on the bulletin board, making verification fail although the result is correct. We have already included these assumptions in the formulation of the verification properties ψ_1, ψ_2 and ψ_{voter}.

In contrast to Eperio, the check ψ_{voter} ensures that no ballot stuffing occurs and that there is a unique ballot for each voter.

Ideally we would like to use Lemma 2 to show the protocol's soundness. To do so, the verification steps must ensure the correct association of elements of the uniquely identifying sort $\mathbb{1}$ with each other sort individually. However, none of the verification steps associate elements of sort $\mathbb{5}$ to elements of sort $\mathbb{1}$. To illustrate that this problem is not a consequence of our protocol modeling, Example 7 shows that the simple protocol as described by Chaum indeed allows the authority to exchange decoy and real ballots at its discretion, as the summands in columns 5, 6, 7, and 8 are not required to be unique, and are chosen freely by the authority.

Example 7. Consider the following two Chaum tables (for simplicity, only columns 1, 5, 6, 7, and 8 are shown).

$A =$

	𝟙	𝟝	𝟞	𝟟	𝟠
#1a − 9343	0000	5555	0000	5555
#1a − 1134	1111	4444	1111	4444
#1b − 7653	2222	3333	2222	3333
#1b − 8584	3333	2222	3333	2222
#2a − 8243	*decoy*	*decoy*	0000	5555
#2a − 5634	*decoy*	*decoy*	1111	4444
#2b − 1253	*decoy*	*decoy*	2222	3333
#2b − 8684	*decoy*	*decoy*	3333	2222

$B =$

	𝟙	𝟝	𝟞	𝟟	𝟠
#1a − 9343	*decoy*	*decoy*	0000	5555
#1a − 1134	*decoy*	*decoy*	1111	4444
#1b − 7653	*decoy*	*decoy*	2222	3333
#1b − 8584	*decoy*	*decoy*	3333	2222
#2a − 8243	0000	5555	0000	5555
#2a − 5634	1111	4444	1111	4444
#2b − 1253	2222	3333	2222	3333
#2b − 8684	3333	2222	3333	2222

The two tables are *not* equal up to permutation of rows. But provided there is a vote corresponding to the voters' expectations (a decoy ballot with the same vote), all verification steps succeed on both tables. △

This weakness can easily be exploited by a malicious authority who can tell voters with real ballots that they are decoy and tell an attacker to buy them, using the votes to influence the result.

To the best of our knowledge, this issue has not been described in the different versions of Chaum's paper. However in some versions of the paper, Chaum proposes an additional check to improve security against vote buying that partially fixes the problem. To prove a decoy ballot is actually a decoy, the authority can reveal its corresponding line numbers in all permuted tables to a voter using a secure channel. The voter can then verify that in all tables his ballot shows up in these lines, and verify that it is actually a decoy when checking ψ_3. Formally, this would imply $\pi_{\{\mathbb{1},\mathbb{5}\}}\sigma_{\mathbb{5}=decoy}R_v \subseteq^{\#}_{\perp} \pi_{\{\mathbb{1},\mathbb{5}\}}P$ for each $v \in \mathcal{V}$ and $P \in \mathcal{P}$.

However this fix does not prevent the authority from telling a voter with a decoy ballot that he has a real ballot (this would correspond to a voter having a *"non-decoy"* value in its column of sort $\mathbb{5}$, while the public table having a *"decoy"* value in the same row) and ϕ is still not satisfied, as the voter's view and the public information cannot be combined. To the best of our knowledge, this latter issue has neither been described in the different versions of Chaum's paper nor elsewhere.

It is clear that the authority cannot provide the voters with a (transferable) proof that their ballot is real together with the ballot, as this would render the decoys useless. A possible fix would be to construct a non-transferable proof. For example, the authority could commit to the line numbers associated to all real ballots, e.g. by inserting them into envelopes and depositing these at a trusted third party. Once the election is over, they can be sent to the voters (using a secure channel) who can then verify they actually received a real ballot. Also, since the numbers are only sent after the election, the voters can create fake receipts by choosing table entries that fit the desired results, making the proof meaningless for anybody else. This step ensures correctness of column $\mathbb{5}$ for all ballots:

$$\psi_{\text{decoy}} := \bigwedge_{v\in\mathcal{V}} \bigwedge_{P\in\mathcal{P}} \pi_{\{\mathbb{1},\mathbb{5}\}}R_v \subseteq^{\#}_{\perp} \pi_{\{\mathbb{1},\mathbb{5}\}}P.$$

E.3.1 Soundness of Chaum's protocol under additional assumptions. We now show that assuming the bulletin board BB and the random draw $\mathbb{D}$ are correct, then the above checks are sufficient to establish soundness of the verification procedure, i.e.

$$\psi_0 \wedge \psi_1 \wedge \psi_2 \wedge \psi_t \wedge \psi_{\text{voter}} \wedge \psi_{\text{decoy}} \Rightarrow \phi. \tag{4}$$

Observe that the tables in $\mathcal{P}$ and the table $\mathbb{D}$ have the same number of rows, and since by ψ_{voter}, $\bigotimes^{\{\mathbb{1}\}}(\mathcal{P} \cup \mathbb{D}) \neq \emptyset$, it must be that for every $P \in \mathcal{P}$, $\pi_{\{V,\mathbb{1}\}}P = \mathbb{D}$.

The check ψ_2 gives us a table $T \in \mathcal{T}$ such that

$$T \in (\bigotimes^{\{\mathbb{1}\}}\{\pi_{\mathbb{1},\mathbb{3}}R \mid R \in Views\}) \cap \mathcal{P}.$$

Since $T \in \mathcal{P}$, check ψ_1 implies that for all $v \in \mathcal{V}$, $\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}R_v \subseteq^{\#} \pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}T$, so that

$$T \in \bigotimes^{\{\mathbb{1}\}}\{\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}R \mid R \in Views\}.$$

The check ψ_{voter} also implies that for each voter $v \in \mathcal{V}$, $\pi_{\{V,\mathbb{1}\}}R_v \subseteq^{\#} \pi_{\{V,\mathbb{1}\}}T$. Hence we also have

$$T \in \bigotimes^{\{\mathbb{1}\}}\{\pi_{\{V,\mathbb{1}\}}R \mid R \in Views\}.$$

Finally the additional check ψ_{decoy} also implies that for all $v \in \mathcal{V}$, $\pi_{\{\mathbb{1},\mathbb{5}\}}R_v \subseteq^{\#} \pi_{\{\mathbb{1},\mathbb{5}\}}T$, and hence

$$T \in \bigotimes^{\{\mathbb{1}\}}\{\pi_{\{\mathbb{1},\mathbb{5}\}}R \mid R \in Views\}.$$

By Lemma 2 we conclude that

$$T \in \bigotimes^{\{\mathbb{1}\}} Views.$$

Finally ψ_t implies that $tal(T) = t$, so that $T \in \mathcal{P}_{con}$. This shows

$$T \in C = \bigotimes^{\{\mathbb{1}\}}(\mathcal{P}_{auth} \cup Views) \cap \mathcal{P}_{con} \neq \emptyset.$$

This also shows that $\bigotimes^{\{\mathbb{1}\}} Views \neq \emptyset$, so we may consider the unique table $\overline{V} \in \mathcal{T}$ given by Lemma 1 such that $\{\overline{V}\} = \min(\bigotimes^{\{\mathbb{1}\}} Views)$. With regards to the second property of ϕ, for any $T \in C$, by ψ_{voter}, $\pi_{\{V,\mathbb{1}\}}T = \pi_{\{V,\mathbb{1}\}}\overline{V}$ and therefore the tables have the same number of rows. Also, since by definition of the merge operator, $\overline{V} \subseteq^{\#} T$, these tables must be equal and hence $tal(T) = tal(\overline{V})$.

E.3.2 Completeness. Similarly to Eperio, it is straightforward to show that $\neg\psi_i$ implies $\neg\phi$ for any property ψ_i in (4).

E.4 Privacy

We analyze privacy in Chaum's protocol following the methodology of Section 5.2. To do so we must first (1) understand the group $S_{\mathcal{V}}$. We fix a table $\bar{C} \in \bigotimes^{Key}(\mathcal{P}_{auth} \cup \tau(Views)) \cap \mathcal{P}_{con}$. Given a voter $v \in \mathcal{V}$, his vote is given by $vote_v = \sigma_{\mathbb{2}=notchecked}\pi_{\{\mathbb{3},\mathbb{4},\mathbb{5}\}}\sigma_{V=v}$. Hence the sort that indicates the voter's vote is $Choice = \{\mathbb{2}, \mathbb{3}, \mathbb{4}, \mathbb{5}\}$, whereas $Fixed = \{\mathbb{1}, V\}$.

$S_{\mathcal{V}}$ is generated by the functions $\tau_{vv'}$ that swap votes between two voters $v, v' \in \mathcal{V}$ that have cast a private vote. In particular $\tau_{vv'}$ swaps the tuples of sort $\{\mathbb{2}, \mathbb{3}, \mathbb{4}, \mathbb{5}\}$ between the subtables $\sigma_{V=v}(\bar{C})$ and $\sigma_{V=v'}(\bar{C})$ by exchanging their votes.

Observe that during the verification procedure analyzed above, the auditors receive table data that amounts to the queries in the set $Q = \{\pi_{\{\mathbb{1},\mathbb{2},\mathbb{4}\}}, \pi_{\{\mathbb{2},\mathbb{3}\}}, \pi_{\{\mathbb{3},\mathbb{4},\mathbb{5}\}}, \pi_{\{\mathbb{1},V\}}\}$. For the second step (2) we prove the following.

Theorem E.1. The elements of sort $\{2, 3, 4, 5\}$ of *not print-audited* ballots are G-indistinguishable under the query set Q and constrains $\mathcal{P}_{con}$, where G is the product of the group of permutations of all such tuples that are marked "notvoted" in column 4, and those that are instead marked "voted".

Proof. To show this, we again first consider the indistinguishability set of elements of sort *Choice* $= \{2, 3, 4, 5\}$ in the table $\bar{C}$ under all queries used in the election process. These are

$$Q = \{\pi_{\{1,2,4\}}, \pi_{\{2,3\}}, \pi_{\{3,4,5\}}, \pi_{\{1,V\}}\}.$$

Applying the same strategy as in Eperio, we examine the indistinguishability set for each query separately.

- $H_1 = Ind_{\bar{C}}(\{\pi_{\{1,2,4\}}\}, \{shape_{\bar{C}}\})$ is the set of tables whose columns of sort $\{1, 2, 4\}$ are equal to those of $\bar{C}$. Since 1-values are unique, tuples of sort $\{2, 3, 4, 5\}$ are G_1 indistinguishable in H_1, where G_1 is the product of the two subgroups J_{voted}, $J_{notvoted}$ of $P_{\bar{C}}(\{M, S\})$. These groups permute tuples of sort $\{2, 3, 4, 5\}$ that contain *"notchecked"* in column 2 and *"voted"*, or *"notvoted"* respectively in column 4.
- $H_2 = Ind_{\bar{C}}(\{\pi_{\{2,3\}}\}, \{shape_{\bar{C}}\})$ is the set of tables whose columns $\{2, 3\}$, indicating whether votes are print audited or not, match those of $\bar{C}$. Since tables are row-wise unordered, tuples of sort $\{2, 3, 4, 5\}$ are completely indistinguishable in H_2, so $G_2 = P_{\bar{C}}(Choice)$.
- $H_3 = Ind_{\bar{C}}(\{\pi_{\{3,4,5\}}\}, \{shape_{\bar{C}}\})$ is the set of tables whose columns $\{2, 3, 4, 5\}$ are equal to those of $\bar{C}$. Similarly to the above case, $G_3 = P_{\bar{C}}(Choice)$.
- Finally, $H_4 = Ind_{\bar{C}}(\{\pi_{\{1,V\}}\}, \{shape_{\bar{C}}\})$. Elements of sort $\{2, 3, 4, 5\}$ are completely indistinguishable in H_4 as this last set is only concerned with columns of sort 1 and V: $G_4 = P_{\bar{C}}(Choice)$.

The result then follows again from Lemma D.1, as $G = G_1$ is a subgroup of each of the groups G_1, G_2, G_3 and G_4. □

As required by (3), it is straightforward to see that $S_{\mathcal{V}}$ is a subgroup of G.

Finally, for the last step (4), let

$$tal = \pi_{\{3\}}\sigma_{4=voted}\sigma_{5=non\text{-}decoy}$$

and recall that

$$\mathcal{P}_{con} = Ind_t^{qu}(\{tal\}) \cap shape_{\mathcal{P}}.$$

For any given $\bar{C} \in \mathcal{P}_{con}$, and any $\tau \in S_{\mathcal{V}}$, we have that $tal(\bar{C}) = tal(\tau(\bar{C}))$ since τ does not change the $\{3, 5\}$ tuples and only permutes marked ones with marked ones. Clearly τ does not change the shape of the table $\bar{C}$. Hence for any $\tau \in S_{\mathcal{V}}$, $\tau(\bar{C}) \in \mathcal{P}_{con}$.

Corollary 2. *By Theorem 1, Chaum's protocol ensures vote privacy as defined in Definition 5.*