A Formal Framework for Quantifying Voter-controlled Privacy[☆,☆☆]

Hugo Jonker[a,*], Sjouke Mauw[a], Jun Pang[a]

[a] *University of Luxembourg, Faculty of Sciences, Technology and Communication*
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg

Abstract

Privacy is a necessary requirement for voting. Without privacy, voters can be forced to vote in specific ways, and the forcing party can check their compliance. But offering privacy does not suffice: if a voter can reduce her privacy, an attacker can force her to do so. In this paper, we distinguish various ways that a voter can communicate with the intruder to reduce her privacy and classify them according to their ability to reduce the privacy of a voter. We develop a framework combining knowledge reasoning and trace equivalences to formally model voting protocols and define voter-controlled privacy. Our framework is quantitative, in the sense that it defines a measure for the privacy of a voter. Therefore, the framework can precisely measure the level of privacy for a voter for each of the identified privacy-reduction classes. The quantification allows our framework to capture receipts that reduce, but not nullify, the privacy of the voter.

Key words: Formal methods, privacy and security, voting protocols

1. Introduction

With the growth and commercialisation of the Internet, people become more and more concerned about their privacy in the digital world. Privacy

[☆]This work was partially supported by a grant of the National Research Fund of Luxembourg.

[☆☆]A preliminary version of this work has appeared in the ARES'09 IEEE proceedings.

[*]Corresponding author

Email addresses: hugo.jonker@uni.lu (Hugo Jonker), sjouke.mauw@uni.lu (Sjouke Mauw), jun.pang@uni.lu (Jun Pang)

has been a fundamental property for systems which provide users e-services, ranging from electronic voting to on-line auctions to health-care.

In voting, vote privacy is the property that an outside observer cannot determine how a voter voted. Although this seems sufficient to ensure privacy, Benaloh and Tuinstra [1] introduce receipt-freeness, which expresses that a voter cannot gain any information to prove to an intruder (someone trying to force the voter to cast a specific vote) that she voted in a certain way. Receipt-freeness aims to prevent vote buying, even when a voter chooses to renounce her privacy. Another stronger notion of privacy is coercion-resistance [2], stating that a voter cannot cooperate with the intruder to prove how she voted. Coercion-resistance also has received considerable attention. These strong notions of privacy actually capture the essential idea that privacy must be enforced by a voting system upon its users, instead offering it.

In the literature, many research efforts have been devoted to ensure privacy properties for (electronic) voting. Several schemes claiming to be receipt-free (e.g. [1, 3, 4]) have been proposed and were later shown to have receipts [5, 6, 7]. Resolving this kind of situation underlines the need for formal methods, which are mathematically based techniques to specify and verify systems. Several formalisations of privacy properties in voting have been proposed. Delaune, Kremer and Ryan [8, 9] develop their formalisation based on observational equivalences of processes in the applied pi calculus, whereas we believe privacy requirements should explicitly model the intruder's knowledge. Based on his knowledge, the intruder can distinguish different traces. Moreover, the privacy of a voter in a voting system also depends on how much knowledge (and when) the voter shares with the intruder. In [10], the tool ProVerif is extended to check some (*not all*) equivalences of [8, 9]. Recently, automatic verification techniques within the applied pi calculus framework have been proposed by Backes, Hriţcu and Maffei [11]. Their approach has its focus on remote electronic voting protocols, and mainly deals with coercion-resistance. The constructive approach of Jonker and De Vink [12] is a logical characterisation of receipt-freeness, but this work is in a less mature state and thus currently only leads itself to high-level analysis. Baskar, Ramanujam and Suresh [13] define a language to specify voting protocols and use an epistemic logic to reason about receipt-freeness. Although it is relatively easy to express privacy properties based on logic of knowledge, it is rather difficult to develop verification techniques within such a logical framework. All the aforementioned approaches only offer qualitative methods to analyse

whether a voting protocol satisfies some privacy property, instead of offering methods to quantify privacy. However, a qualitative approach leaves several possibilities for an intruder to reduce voter privacy. Examples include forcing a voter *not* to vote for a certain candidate, or determining at which end of the political spectrum was voted. We believe that computing the exact possible choices of a voter is the only way to detect such attacks. In our view, any privacy reductor is a receipt, not just those that nullify privacy.

In this paper we consider the possibilities for a voter to reduce her privacy, which we call *voter-controlled privacy*. The voter can *subjectively* provide the intruder information, to prove that she has voted in a certain way in order to sell her vote. The intruder tries to find out whether the voter did vote as she said, based on his knowledge gained by observing the election and/or communicating with the voter. The privacy of any voting system largely depends on the possibilities afforded to the intruder to communicate with a voter. Voting literature [14, 15, 5, 4, 16] uses the following types of information-hiding techniques to ensure privacy: mixnets [16], blind signatures [14], homomorphic encryption [5]. Furthermore, private untappable one-way channels (voter to authority or authority to voter) or two-way channels are often assumed [15, 5, 4]. The ability to break privacy relies on the information acquired by the intruder. Such channels provide a means to keep information from the intruder and thus help privacy. However, information can be forwarded to the intruder by the voter. And vice versa, a voter could use information supplied by the intruder (which then serves as a witness). These observations lead us to distinguish various independent ways that a voter can communicate with the intruder to rescind her privacy and classify them according to their ability to reduce the privacy of a voter. These classes are ordered according to privacy-reducing ability. The goal of this paper is to provide a method for quantifying privacy of voters within this ordering.

As made clear by Benaloh and Tuinstra [1], voter privacy depends on which knowledge the voter shares with the intruder, and at what time during the protocol this knowledge is shared. (This is captured by the ordering of the conspiracy classes in our paper.) Therefore, we develop a framework combining knowledge reasoning along the lines of [12, 13] and trace equivalences of [8, 9] to formally model voting protocols and define vote privacy for the voters. This enables us to describe the knowledge of the intruder, as well as of other entities in a voting protocol, at any point during execution of the system. To distinguish execution traces with respect to the current knowledge of the intruder, we adapt the reinterpretation function proposed

by Garcia et al. [17]. The framework is quantitative, in the sense that it precisely measures privacy for any voter conspiring like one of the conspiracy classes. This is achieved by establishing per-voter choice groups, which take different communication mechanisms between voters and the intruder into account, along the lines of anonymity groups as introduced by Mauw, Verschuren and De Vink [18] and Chothia et al. [19]. Thus, the framework captures attacks which reduce, but not nullify, privacy. Such attacks have not been identified and dealt with in other formal approaches.

Contribution. Our contribution in this paper is twofold. First, we introduce a new formal framework combining knowledge reasoning and trace equivalences to model and reason about voting protocols. Second, we provide a quantitative definition of voter-controlled privacy. It allows for detection of subtle privacy attacks on voters. The different cooperation modes of a voter and the intruder are captured by conspiracy ordering.

Outline. The rest of this paper is organised as follows. In Section 2, we define the setting of our framework. In Section 3, a formal framework to model voting systems and its operational semantics is presented, which is followed by a quantitative definition of privacy in voting in Section 4, based on knowledge reasoning and trace equivalences. In Section 5, we classify how a voter can cooperate with the intruder and in Section 6, we formally define how to measure privacy for these classes of conspiring behaviour. We illustrate our framework by two examples in Section 7 and discuss the power of our framework by identifying some new attacks which can only be detected by a quantitative approach in Section 8. Finally, Section 9 concludes the paper and lists some future works.

2. Setting

This paper restricts itself to studying privacy of a voting system at the protocol level. The setting for our approach is as follows.

2.1. Election type

We call the type of elections studied in this paper `1v1v` (short for "one voter, one vote"). `1v1v` elections have the following properties:

- There is a set of voters $\mathcal{V}$, a set of candidates $\mathcal{C}$, and a set of voting authorities Aut. We denote the set of all agents as $Agents = \mathcal{V} \cup Aut$.

- Each voter $v \in \mathcal{V}$ is entitled to cast one vote for one candidate $c \in \mathcal{C}$.
- A public-key infrastructure for the public key $pk(a)$ and private keys $sk(a)$ of agent $a \in Agents$.
- All votes have equal weight.
- The set of received ballots precisely determines the election result, and is published after the elections.
- The candidate preferred by a voter is independent of the voting system, which implies that the voter's choice can be given a priori.

Voting systems often distinguish between various phases, such as a registration phase, a voting phase and a counting phase. The below framework expresses phases as a synchronisation of parties.

2.2. Primitives

The following cryptographic and communication primitives are used in the framework.

- *Nonces* are freshly generated random values.
- *Encryption* of a plain text φ with key k is denoted as $\{\varphi\}_k$. The resulting cipher text can be decrypted using the inverse key k^{-1}. We distinguish symmetric encryption, where $k^{-1} = k$ and asymmetric encryption such as public key encryption, where $k = pk(a)$ (or $sk(a)$) and $k^{-1} = sk(a)$ (or $pk(a)$), for some agent a.
- *Public channels* are communication channels that reveal everything about communication: sender, intended receiver and message.
- *Anonymous channels* are communication channels that hide the sender of a message. Other than that, anonymous channels reveal intended receiver and message.
- *Untappable channels* are communication channels that are completely private. Only the sender and receiver learn the communicated message, and only they are aware that a communication occurred at all. Untappable channels can be either directed (i.e. a one-way communication channel) or bi-directional (i.e. a two-way communication channel). Section 5 discusses the consequences of untappable channels further.

2.3. The intruder model

We consider a standard Dolev-Yao intruder [20] throughout the paper. This intruder model can be characterised as follows:

- perfect cryptography assumption: no encrypted message can be opened without the decryption key;
- network control:
  - all messages sent on public and on anonymous channels are received by the intruder;
  - agents can only receive messages from the intruder.

  Thus, the intruder can block or modify any message communicated over a public or anonymous channel as he sees fit. Note that untappable channels are not under intruder control.

3. A framework for modelling voting systems

In this section, we develop a formal syntax and provide semantics for expressing the communication behaviour of voting systems in terms of *agents*. We first specify syntax and derivation of terms. This is followed by the syntax of communication events, and the syntax of processes (which specify the order in which an agent executes events). Together, this enables us to syntactically express voting systems. Next, semantics is given at two levels: the level of individual agents, and the level of the voting system as a whole.

As a consequence of the assumption that the way voters vote is independent of the election process, the relation between voters and their chosen candidates can be given a priori. This is captured by the relation $\gamma\colon \mathcal{V} \to \mathcal{C}$, which specifies for each voter $v \in \mathcal{V}$ which candidate $\gamma(v) \in \mathcal{C}$ she votes for. To deconstruct tuples, we use projection functions $\pi_i, i > 0$ which return the i^{th} component of a tuple.

3.1. Syntax of the framework

We first define the syntax of terms, which is followed by the syntax of agents. Next, the syntactical representation of a voting system is defined as a function associating a state with each agent.

3.1.1. Terms

The terms communicated in a voting system are built up from variables from the set Vars, candidates from the set $\mathcal{C}$, random numbers from the set *Nonces*, and cryptographic keys from the set *Keys*. The set of keys of a particular agent a is given by $Keys_a$. These basic terms can be composed through pairing ((φ_1, φ_2)) and encryption ($\{\varphi\}_k$).

Definition 1 (terms)**.** *Let* Vars *be a set of variables, containing at least the variable* vc*, let Agents be a set of agents, let* $\mathcal{C}$ *be a set of candidates, let Nonces be a set of nonces, and let Keys be a set of keys, ranged over by* var, a, c, n *and* k*, respectively. The class Terms of terms, ranged over by* φ*, is given by the BNF*

$$\varphi ::= \mathsf{var} \mid a \mid c \mid n \mid k \mid (\varphi_1, \varphi_2) \mid \{\varphi\}_k.$$

Syntactical equivalence of terms φ_1, φ_2 *is denoted as* $\varphi_1 = \varphi_2$*. A term is called* open *if it contains variables and* closed *if it contains no variables. The set of variables of an open term* φ *is given by* $\mathrm{fv}(\varphi)$*.*

Terms encrypted with k *can be decrypted using the inverse key* k^{-1}*. For symmetric encryption,* $k^{-1} = k$*, whereas in asymmetric encryption,* $pk(a)$ *denotes the public key and* $sk(a)$ *the corresponding secret key of agent* a*. Signing is denoted as encryption with the secret key.*

Example (terms). The encryption of a nonce n with the public key of agent a is denoted as $\{n\}_{pk(a)}$. A pair of this encrypted nonce with itself, unencrypted, results in the term $(\{n\}_{pk(a)}, n)$. Note that $((\{n\}_{pk(a)}, n), n) \neq (\{n\}_{pk(a)}, (n, n))$, as the terms are syntactically different.

Variables represent unspecified terms, such as a voter's choice. The voter's choice is represented by the variable vc until instantiated.

Definition 2 (term substitution)**.** *We introduce a variable mapping relation* $\mapsto$*, of type* $\mathsf{Vars} \to$ *Terms. The substitution of a single variable* var *by term* φ *is denoted as* $\mathsf{var} \mapsto \varphi$*. Application of substitution* σ *to term* φ *is denoted as* $\sigma(\varphi)$*. The domain of substitution* σ*, notation* $\mathrm{dom}(\sigma)$*, is the set of variables for which* σ *specifies a substitution. The range of a substitution* σ*, notation* $\mathrm{rng}(\sigma)$*, is defined as* $\{\sigma(\mathsf{var}) \mid \mathsf{var} \in \mathrm{dom}(\sigma)\}$*. The composition of two substitutions* σ' *after* σ *is denoted by* $\sigma' \circ \sigma$*. The empty substitution is denoted as* $\emptyset$*.*

Agents communicating terms can expect to receive a term with a certain structure, for example, a term encrypted with the agent's public key ($\{\varphi\}_{pk(a)}$). This is expressed by specifying an open term for the receiving agent (such as $\{\mathsf{var}\}_{pk(a)}$), that serves as a template for the expected term.

Definition 3 (term matching)**.** *A closed term φ_1 matches a term φ_2, if there is a substitution σ, such that* $\mathrm{dom}(\sigma) = \mathrm{fv}(\varphi_2)$ *and* $\sigma(\varphi_2) = \varphi_1$*. We denote this as*

$$\mathrm{match}(\varphi_1, \varphi_2, \sigma) \;\equiv\; \sigma(\varphi_2) = \varphi_1 \wedge \mathrm{dom}(\sigma) = \mathrm{fv}(\varphi_2).$$

Lemma 1. (unique substitution). *Given a closed term φ_1 and a term φ_2, there is at most one substitution σ such that* $\mathrm{match}(\varphi_1, \varphi_2, \sigma)$*.*

Proof. (Proof by contradiction) Suppose that for two terms φ_1, φ_2 there exist σ_1, σ_2 such that:

- $\mathrm{match}(\varphi_1, \varphi_2, \sigma_1)$ and $\mathrm{match}(\varphi_1, \varphi_2, \sigma_2)$ both hold, while
- $\sigma_1 \neq \sigma_2$, $\sigma_1(\varphi_2) = \sigma_2(\varphi_2) = \varphi_1$, and $\mathrm{dom}(\sigma_1) = \mathrm{dom}(\sigma_2) = \mathrm{fv}(\varphi_2)$.

As $\mathrm{dom}(\sigma_1) = \mathrm{dom}(\sigma_2)$ and $\sigma_1 \neq \sigma_2$, we have $\mathrm{rng}(\sigma_1) \neq \mathrm{rng}(\sigma_2)$. Thus, there must be at least one variable $\mathsf{var} \in \mathrm{dom}(\sigma_1)$ such that $\sigma_1(\mathsf{var}) \neq \sigma_2(\mathsf{var})$. But since $\mathrm{dom}(\sigma_1) = \mathrm{fv}(\varphi_2)$, $\mathsf{var} \in \mathrm{fv}(\varphi_2)$. Thus $\sigma_1(\varphi_2) \neq \sigma_2(\varphi_2)$, which is a contradiction. □

Example (variable matching and substitution). Consider the term of the previous example, $\{n\}_{pk(a)}$. In Table 1, we compare this term to several other terms to see if they match under the stated substitution. In Table 1, we assume that $n \neq n'$ and $pk(a) \neq k$.

φ	σ	$\mathrm{match}(\{n\}_{pk(a)}, \varphi, \sigma)$
var	$\mathsf{var} \mapsto \{n\}_{pk(a)}$	true
$\{\mathsf{var}\}_{pk(a)}$	$\mathsf{var} \mapsto n$	true
$\{n\}_{pk(a)}$	$\emptyset$	true
$\{n'\}_{pk(a)}$	any	false
$\{n\}_k$	any	false

Table 1: Example: matching and substitution of terms

A term φ may be derived from a set of terms K (notation $K \vdash \varphi$) if it is an element of K or if it can be derived by repeated application of the following rules:

$$\begin{array}{lll}
\text{(pairing)} & K \vdash \varphi_1,\ K \vdash \varphi_2 & \Longrightarrow\ K \vdash (\varphi_1, \varphi_2) \\
\text{(left)} & K \vdash (\varphi_1, \varphi_2) & \Longrightarrow\ K \vdash \varphi_1 \\
\text{(right)} & K \vdash (\varphi_1, \varphi_2) & \Longrightarrow\ K \vdash \varphi_2 \\
\text{(encryption)} & K \vdash \varphi,\ K \vdash k & \Longrightarrow\ K \vdash \{\varphi\}_k \\
\text{(decryption)} & K \vdash \{\varphi\}_k,\ K \vdash k^{-1} & \Longrightarrow\ K \vdash \varphi
\end{array}$$

For instance, the decryption rule enables an agent a to decrypt $\{n\}_{pk(a)}$, as he posses the key $sk(a)$. An agent's knowledge is a set of terms closed under derivability. Closure of a set K under derivability is defined as $\overline{K} = \{\varphi \mid K \vdash \varphi\}$.

Example (derivability and knowledge). From a knowledge set K containing term n, we can derive the following terms: n, (n, n), $((n, n), n)$, $(n, (n, n))$, and so on. If K also contains a key k, we can additionally derive terms such as $\{n\}_k$, $(n, \{n\}_k)$, (n, k), (k, n), ...

3.1.2. Agents

Terms are communicated between agents. Communication of a term is an event. We distinguish public, anonymous, and untappable communication channels. Hence, there are distinct events for each type of channel. Furthermore, we note that any election process inherently has multiple phases. As such, synchronisation between the election officials must be inherent in voting systems. For the framework to support this, we introduce a phase synchronisation event.

Definition 4 (events)**.** *The class Ev of communication events, ranged over by ev, is given by:*

$$\begin{array}{rl}
Ev = & \{s(a, a', \varphi), r(a, a', \varphi), as(a, a', \varphi), ar(a', \varphi), us(a, a', \varphi), ur(a, a', \varphi), \\
& \quad ph(i) \mid a, a' \in Agents, \varphi \in Terms, i \in \mathbb{N}\ \},
\end{array}$$

where s, r, as, ar, us, ur denote sending and receiving over public, anonymous and untappable channels, respectively. Finally, $ph(i)$ is the event denoting that an agent is ready to execute phase transition i.

The subclass of send events is denoted as:

$$Ev_{snd} = \{s(a, a', \varphi), as(, a', \varphi), us(a, a', \varphi) \mid a, a' \in Agents, \varphi \in Terms\}.$$

Similarly, the subclass of receive events is denoted as:

$$Ev_{rcv} = \{r(a, a', \varphi), ar(a', \varphi), ur(a, a', \varphi) \mid a, a' \in Agents, \varphi \in Terms\}.$$

The subclass of phase events is denoted as $Ev_{ph} = \{ph(i) \mid i \in \mathbb{N}\}$.

Variable substitution is extended straightforwardly to events, by replacing all substituted variables in the term of one event. This is denoted as $\sigma(ev)$ for an event ev and a substitution σ. The function fv is similarly extended, by application to the term of an event. The set of free variables of an event ev is thus given by $\mathrm{fv}(ev)$.

The behaviour of an agent is determined by the order in which events occur. This order is defined by the agent's process, as follows.

Definition 5 (processes)**.** *The class Processes of processes, ranged over by* P*, is defined as follows.*

$$P \quad ::= \quad \delta \mid ev.P \mid P_1 + P_2 \mid P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2 \mid ev.X(\varphi_1, \ldots, \varphi_n).$$

Here, δ *denotes a deadlock. A process can be preceded by an event* $(ev.P)$*. Furthermore, a process can be a choice between two alternative processes, either a non-deterministic choice* $(P_1 + P_2)$ *or a conditional choice* $(P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2)$*. If* φ_1 *is syntactically equal to* φ_2*, the process behaves as* P_1*; otherwise, the process behaves as* P_2*. Finally, we have guarded recursion of processes. We assume a class of process variables, which is ranged over by* X*. For every process variable* X*, with arity* n*, there is a defining equation of the form* $X(\mathsf{var}_1, \ldots, \mathsf{var}_n) = P$*, with the syntactic requirement that the free variables of* P *(as defined below) are precisely* $\mathsf{var}_1, \ldots, \mathsf{var}_n$*.*

Example (processes). The following are examples of processes.

$$\begin{array}{l} \delta \\ ph(1).\delta \\ ph(2).\delta \lhd \{\varphi\}_k = \{\varphi'\}_k \rhd ph(1).\delta \\ ph(1).X(\varphi_1, \{\varphi_2\}_k, n) \end{array}$$

Without loss of generality, we assume a naming convention such that all free variables in the defining equation of a process variable have globally unique names. This limits the scope of such variables to that defining equation.

Substitutions are extended to processes. The substitution σ applied to process P is denoted as $\sigma(P)$ and is defined as follows.

$$\sigma(P) = \begin{cases} \delta & \text{if } P = \delta \\ \sigma(ev).\sigma(P') & \text{if } P = ev.P' \\ \sigma(P_1) + \sigma(P_2) & \text{if } P = P_1 + P_2 \\ \sigma(P_1) \lhd \sigma(\varphi_1) = \sigma(\varphi_2) \rhd \sigma(P_2) & \text{if } P = P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2 \\ \sigma(ev).Y(\sigma(\varphi_2), \ldots, \sigma(\varphi_n)) & \text{for fresh } Y(\mathsf{var}_1, \ldots, \mathsf{var}_n) = \sigma(P') \\ & \text{if } P = ev.X(\varphi_1, \ldots, \varphi_n) \ \wedge \ X(\mathsf{var1}, \ldots, \mathsf{var}_n) = P' \end{cases}$$

Note that for guarded recursion (i.e. $ev.X(\varphi_1, \ldots, \varphi_n)$), the substitution σ is applied to the invoked process too (i.e. to P, if $X(\mathsf{var}_1, \ldots, \mathsf{var}_n) = P$). This is done by introducing a new defining equation for a fresh process variable Y. The defining equation is of the form $Y(\mathsf{var}_1, \ldots, \mathsf{var}_n) = P'$, where P' is the substitution σ applied to process P, so $Y(\mathsf{var}_1, \ldots, \mathsf{var}_n) = \sigma(P)$.

The function fv is also extended to processes. Note that receiving a term binds the received variables, hence receive actions reduce the number of free variables. Furthermore, remark that in guarded recursion all free variables of the defined process are bound by the invocation (due to the syntactic requirement). Thus, the only free variables of an invocation are the free variables of the argument list.

$$\mathrm{fv}(P) = \begin{cases} \emptyset & \text{if } P = \delta \\ \mathrm{fv}(P') & \text{if } P = ev.P' \wedge ev \in Ev_{ph} \\ \mathrm{fv}(P') \cup \mathrm{fv}(ev) & \text{if } P = ev.P' \wedge ev \in Ev_{snd} \\ \mathrm{fv}(P') \setminus \mathrm{fv}(ev) & \text{if } P = ev.P' \wedge ev \in Ev_{rcv} \\ \mathrm{fv}(P_1) \cup \mathrm{fv}(P_2) & \text{if } P = P_1 + P_2 \\ \mathrm{fv}(P_1) \cup \mathrm{fv}(\varphi_1) \cup \mathrm{fv}(\varphi_2) \cup \mathrm{fv}(P_2) & \text{if } P = P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2 \\ \mathrm{fv}(\varphi_1) \cup \cdots \cup \mathrm{fv}(\varphi_n) & \text{if } P = ev.X(\varphi_1, \ldots, \varphi_n) \\ & \quad \wedge \ ev \in Ev_{ph} \\ \mathrm{fv}(\varphi_1) \cup \cdots \cup \mathrm{fv}(\varphi_n) \cup \mathrm{fv}(ev) & \text{if } P = ev.X(\varphi_1, \ldots, \varphi_n) \\ & \quad \wedge \ ev \in Ev_{snd} \\ \mathrm{fv}(\varphi_1) \cup \cdots \cup \mathrm{fv}(\varphi_n) \setminus \mathrm{fv}(ev) & \text{if } P = ev.X(\varphi_1, \ldots, \varphi_n) \\ & \quad \wedge \ ev \in Ev_{rcv} \end{cases}$$

Variables used in a defining equation which do not appear in the argument list of the equation must be bound by their occurrence. This means that such variables may only occur in events which bind variables, i.e. events $\in Ev_{rcv}$.

An agent's state is a combination of behaviour, i.e. the order in which events are executed (as determined by a process), and knowledge (a set of terms) as follows.

Definition 6 (agent state)**.** *The state of an agent a is given by the agent's knowledge knw_a and its process P_a. Thus, agent state $st \in Agstate$ is a tuple of knowledge and a process:*

$$Agstate = \mathcal{P}(Terms) \times Processes.$$

A voting system specifies, for each agent, its state. Hence, a voting system is a mapping from agent to states.

Definition 7 (voting system)**.** *A voting system is a system that specifies a state for each agent a. The class of voting systems, VotSys, is defined as $VotSys = Agents \rightarrow Agstate$. We denote the state assigned by voting system $\mathcal{VS} \in VotSys$ to agent a as $\mathcal{VS}(a) = (knw_a, P_a)$. knw_a is the knowledge of agent a, and P_a describes its behaviour.*

A voting system $\mathcal{VS} \in VotSys$ may be instantiated with voter choices, as given by choice function $\gamma \colon \mathcal{V} \rightarrow \mathcal{C}$. This instantiation is denoted as $\mathcal{VS}^{\gamma}$, which, for each voter, substitutes the voter choice variable vc *by the choice specified by γ in her process, as follows.*

$$\mathcal{VS}^{\gamma}(a) = \begin{cases} \mathcal{VS}(a) & \text{if } a \notin \mathcal{V} \\ \big(\ \pi_1(\mathcal{VS}(a)), \pi_2(\sigma(\mathcal{VS}(a)))\ \big) & \text{if } a \in \mathcal{V} \wedge \sigma = \mathsf{vc} \mapsto \gamma(a) \end{cases}$$

Recall that π_i denotes an extraction function that extracts the i^{th} component from a tuple.

Example (voting system). In Table 2 we specify a voting system $\mathcal{VS}_0$ with two voters *v1*, *v2* and one authority T. In $\mathcal{VS}_0$, voter *v1* sends her signed vote, encrypted with T's public key, to counter T. Voter *v2* does the same for her vote. Note that these votes are specified as vc, until instantiated by a choice function. The counter receives these votes in variables var_1 and var_2, respectively. The initial knowledge of a voter consists of her private key and the public key of T. As there are no voter-to-voter interactions, we omit the public keys of other voters in the initial knowledge of a voter in this example. In addition to $\mathcal{VS}_0$, we also show $\mathcal{VS}_o^{\gamma_1}$, where $\gamma_1(v1) = c1$ and $\gamma_1(v2) = c2$.

$$
\begin{array}{ll}
\mathcal{VS}_0(v1) & = (\{pk(T), sk(v1)\},\ s(v1, T, \{\{\mathsf{vc}\}_{sk(v1)}\}_{pk(T)})\ .\ \delta) \\
\mathcal{VS}_0(v2) & = (\{pk(T), sk(v2)\},\ s(v2, T, \{\{\mathsf{vc}\}_{sk(v2)}\}_{pk(T)})\ .\ \delta) \\
\mathcal{VS}_0(T) & = (\{pk(v1), pk(v2), sk(T)\}, \\
 & \quad r(v1, T, \{\{\mathsf{var}_1\}_{sk(v1)}\}_{pk(T)})\ .\ r(v2, T, \{\{\mathsf{var}_2\}_{sk(v2)}\}_{pk(T)})\ .\ \delta)
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{VS}_0^{\gamma_1}(v1) & = (\{pk(T), sk(v1)\},\ s(v1, T, \{\{c1\}_{sk(v1)}\}_{pk(T)})\ .\ \delta) \\
\mathcal{VS}_0^{\gamma_1}(v2) & = (\{pk(T), sk(v2)\},\ s(v2, T, \{\{c2\}_{sk(v2)}\}_{pk(T)})\ .\ \delta) \\
\mathcal{VS}_0^{\gamma_1}(T) & = \mathcal{VS}_0(T)
\end{array}
$$

Table 2: Example of voting system $\mathcal{VS}_0$ and $\mathcal{VS}_0^{\gamma_1}$ (where $\gamma_1(v1) = c1$ and $\gamma_1(v2) = c2$)

3.2. Semantics of the framework

The operational semantics of a voting system is defined by a number of derivation rules of the form

$$\frac{p_1 \ldots p_n}{S \xrightarrow{e} S'}.$$

This expresses that if the system is in state S and if the premises p_1 to p_n are satisfied, the system may perform event e and continue in state S' [21]. The operational semantics is defined in two layers. First, the semantics of individual agents is defined. Next we define the semantics of a voting system based on the semantics of individual agents. The operational semantics of a voting system can be seen as the parallel composition of all agents.

3.2.1. Agent semantics

The semantics of agents describes the effect of the events on the agent state. Recall that agent state is defined as a tuple containing a knowledge set and a process: $Agstate = \mathcal{P}(Terms) \times Processes$.

In our assumed intruder model, each tappable communication by an agent is a communication with the intruder. Hence, the semantic rules below take the intruder's knowledge into account. The states considered below thus consist of a tuple of intruder knowledge K_I and agent state. In the rules below, events may involve $a, x \in Agents$, which we omit from the premise of the rules.

There are some restrictions on the terms that may occur in an event. A term φ occurring in a send event must be closed ($\mathrm{fv}(\varphi) = \emptyset$) at the moment of sending. A term occurring in a receive event can specify the structure

of the term to be received. There is a limitation: a receiving agent a can only specify the structure up to the extent of his knowledge knw_a. This is captured by the readable predicate $\mathrm{Rd}\colon \mathcal{P}(\mathit{Terms}) \times \mathit{Terms}$ (inspired by [22]).

$$\mathrm{Rd}(knw_a, \varphi) \equiv \begin{cases} \mathit{true} & \text{if } \varphi \in \mathsf{Vars} \\ knw_a \vdash \varphi & \text{if } \varphi \in \mathcal{C} \cup \mathit{Nonces} \cup \mathit{Keys} \cup \mathit{Agents} \\ knw_a \vdash \varphi \vee (knw_a \vdash k^{-1} \wedge \mathrm{Rd}(knw_a, \varphi_1)) & \text{if } \varphi = \{\varphi_1\}_k \\ (\mathrm{Rd}(knw_a, \varphi_2) \wedge \mathrm{Rd}(knw_a \cup \{\varphi_2\}, \varphi_1)) \vee (\mathrm{Rd}(knw_a, \varphi_1) \wedge \mathrm{Rd}(knw_a \cup \{\varphi_1\}, \varphi_2)) & \text{if } \varphi = (\varphi_1, \varphi_2) \end{cases}$$

An agent a may use term φ as a template for a receive event, when φ is either a variable or a constant term derivable from knw_a. When φ is an encryption, it may be used as a template if the agent knows the inverse key, or if the agent knows the key and can read the encrypted message. For pairing, we take into account that either side of the pair may contain information (e.g. a key) that is necessary to read the other side.

Example (readability of terms). Below, we illustrate the readability function for several terms and knowledge sets (assuming symmetric encryption).

φ	knw	$\mathrm{Rd}(knw, \varphi)$
(n, k)	$\emptyset$	*false*
$\{\mathsf{var}\}_k$	$\emptyset$	*false*
$\{n\}_k$	$\{n\}$	*false*
$(k, \{\mathsf{var}\}_k)$	$\emptyset$	*false*

φ	knw	$\mathrm{Rd}(knw, \varphi)$
var	$\emptyset$	*true*
(n, k)	$\{n, k\}$	*true*
$\{\mathsf{var}\}_k$	$\{k\}$	*true*
$\{n\}_k$	$\{n, k\}$	*true*
$(k, \{\mathsf{var}\}_k)$	$\{k\}$	*true*
$(\mathsf{var}_1, \{\mathsf{var}_2\}_{\mathsf{var}_1})$	$\emptyset$	*true*

public send. An agent may send a closed term φ if and only if the agent knows the term (note that variables do not capture knowledge).

$$\frac{knw_a \vdash \varphi \qquad \mathrm{fv}(\varphi) = \emptyset}{(K_I, knw_a, s(a, x, \varphi).P) \xrightarrow{s(a,x,\varphi)} (K_I \cup \{\varphi\}, knw_a, P)}$$

public receive. A receive event specifies an open, readable term φ. By receiving a matching term φ', the unassigned variables in φ are assigned a value by the unique substitution σ such that $\text{match}(\varphi', \varphi, \sigma)$.

$$\frac{K_I \vdash \varphi' \qquad \text{fv}(\varphi') = \emptyset \qquad \text{match}(\varphi', \varphi, \sigma) \qquad \text{Rd}(knw_a, \varphi)}{(K_I, knw_a, r(x, a, \varphi).P) \xrightarrow{r(x,a,\varphi')} (K_I, knw_a \cup \{\varphi'\}, \sigma(P))}$$

anonymous send. The anonymous send semantics follows directly from the above public send semantics by replacing the send event $s(x, y, \varphi)$ with event $as(x, y, \varphi)$.

$$\frac{knw_a \vdash \varphi \qquad \text{fv}(\varphi) = \emptyset}{(K_I, knw_a, as(, x, \varphi).P) \xrightarrow{as(,x,\varphi)} (K_I \cup \{\varphi\}, knw_a, P)}$$

anonymous receive. An anonymous receive is largely equal to the public receive, except that the event omits information about the sender.

$$\frac{K_I \vdash \varphi' \qquad \text{fv}(\varphi') = \emptyset \qquad \text{match}(\varphi', \varphi, \sigma) \qquad \text{Rd}(knw_a, \varphi)}{(K_I, knw_a, r(a, \varphi).P) \xrightarrow{ar(a,\varphi')} (K_I, knw_a \cup \{\varphi'\}, \sigma(P))}$$

untappable send. An untappable send is a send event which happens outside intruder control or even intruder awareness. It thus limits the power of the intruder. Note that the intruder does not learn the communicated term for an untappable send event.

$$\frac{knw_a \vdash \varphi \qquad \text{fv}(\varphi) = \emptyset}{(K_I, knw_a, us(a, x, \varphi).P) \xrightarrow{us(a,x,\varphi)} (K_I, knw_a, P)}$$

untappable receive. An untappable receive is the receiving dual of the untappable send. Thus, it occurs outside intruder control or awareness, and represents a limit on the power of the intruder. Note that, at this level of the semantics, the origin of the term φ' is not specified. The origin of this term is specified at the system level, where untappable receive is synchronised with untappable send.

$$\frac{\text{fv}(\varphi') = \emptyset \qquad \text{match}(\varphi', \varphi, \sigma) \qquad \text{Rd}(knw_a, \varphi)}{(K_I, knw_a, ur(x, a, \varphi).P) \xrightarrow{ur(x,a,\varphi')} (K_I, knw_a \cup \{\varphi'\}, \sigma(P))}$$

phase synchronisation. The events of the set Ev_{ph} are intended to synchronise agents in the system. At the agent level, these events have little impact, as evidenced by the following operational semantic rule.

$$\frac{ev \in Ev_{ph}}{(K_I, knw_a, ev.P) \xrightarrow{ev} (K_I, knw_a, P)}$$

non-deterministic choice. An agent that can execute a non-deterministic choice may choose any of the alternatives, as follows.

$$\frac{(K_I, knw_a, P_1) \xrightarrow{ev} (K_I', knw_a', P_1')}{(K_I, knw_a, P_1 + P_2) \xrightarrow{ev} (K_I', knw_a', P_1')}$$

$$\frac{(K_I, knw_a, P_2) \xrightarrow{ev} (K_I', knw_a', P_2')}{(K_I, knw_a, P_1 + P_2) \xrightarrow{ev} (K_I', knw_a', P_2')}$$

conditional choice. A conditional choice is a choice between two processes, based upon syntactical comparison of two terms. While these two terms may be open terms in the agent's specification, upon execution, we require that these terms are closed.

$$\frac{(K_I, knw_a, P_1) \xrightarrow{ev} (K_I', knw_a', P_1') \qquad \mathrm{fv}(\varphi_1) = \emptyset}{(K_I, knw_a, P_1 \lhd \varphi_1 = \varphi_1 \rhd P_2) \xrightarrow{ev} (K_I', knw_a', P_1')}$$

$$\frac{(K_I, knw_a, P_2) \xrightarrow{ev} (K_I', knw_a', P_2') \qquad \varphi_1 \neq \varphi_2 \qquad \mathrm{fv}(\varphi_1) = \mathrm{fv}(\varphi_2) = \emptyset}{(K_I, knw_a, P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2) \xrightarrow{ev} (K_I', knw_a', P_2')}$$

guarded recursion. An invocation of process variable X with argument list $\varphi_1, \ldots, \varphi_n$ can be executed by agent a if the corresponding process in the defining equation can execute, under the specified arguments.

$$\frac{\begin{array}{c}(K_I, knw_a, \sigma(P)) \xrightarrow{ev} (K_I', knw_a', P') \\ X(\mathsf{var}_1, \ldots, \mathsf{var}_n) = P \qquad \sigma = \mathsf{var}_1 \mapsto \varphi_1 \circ \cdots \circ \mathsf{var}_n \mapsto \varphi_n\end{array}}{(K_I, knw_a, X(\varphi_1, \ldots, \varphi_n)) \xrightarrow{ev} (K_I', knw_a', P')}$$

3.2.2. System semantics

The operational semantics of voting systems describe how the state of a voting system changes due to the interactions of its agents. The state of a voting system is given by the intruder knowledge and the state of each agent.

Definition 8 (state of a voting system)**.** *The state of a voting system is a tuple of intruder knowledge and a mapping of agents to agent states (recall that VotSys = Agents → Agstate), as follows.*

$$State = \mathcal{P}(Terms) \times VotSys.$$

The knowledge and current process for each agent are given by VotSys. We denote the attribution of state (knw_a, P_a) *to agent* a *as* $a \mathbin{@} (knw_a, P_a)$*. The current state of agent* a *in system state* (K_I, S) *is denoted as* $a \mathbin{@} (knw_a, P_a) \in S$*. The initial state of voting system* $\mathcal{VS}$ *with respect to choice function* γ *is* $(K_I^0, \mathcal{VS}^\gamma)$*, for initial intruder knowledge* K_I^0*.*

Typically, the initial intruder knowledge contains public keys of all agents, compromised keys etc.

The operational semantics of voting systems in the context of a Dolev-Yao intruder (limited to public and anonymous channels) is given below. The semantic rules give rise to a labelled transition system, with labels denoting the events. Untappable communication is modelled as synchronous communication, hence the *us* and *ur* events are replaced by *uc* events (denoting untappable communication) in the set of labels *Labels* of the transition system:

$$\begin{aligned} Labels \;=\; & \{uc(a, a', \varphi) \mid a, a' \in Agents \wedge \varphi \in Terms\} \cup Ev \setminus \\ & \{ur(a, a', \varphi), ur(a, a', \varphi) \mid a, a' \in Agents \wedge \varphi \in Terms\}. \end{aligned}$$

Both untappable communications and phase synchronisation are synchronous events by more than one agent. The other events are executed without synchronising with other agents. We distinguish the non-synchronous events as Ev_{nosync}, which is defined as follows.

$$\begin{aligned} Ev_{nosync} \;=\; & \{s(a, a', \varphi), r(a, a', \varphi), as(, a', \varphi), ar(a', \varphi) \\ & \mid a, a' \in Agents, \varphi \in Terms\}. \end{aligned}$$

The below system semantics uses the above agent semantics to define the dynamic behaviour of the system. The rules below may involve agents $a, b \in Agents$, which we omit from the premises of the rules. Note that the premise of the rules involves agent state transitions (a three-tuple of intruder knowledge, agent knowledge and agent process), and may specify restrictions on the system state (a mapping of agents to agent states).

non-synchronous events. The operational semantics for public and for anonymous send as well as read events is given by the following rule.

$$\frac{(K_I, knw_a, P) \xrightarrow{ev} (K_I', knw_a', P') \qquad ev \in Ev_{nosync} \qquad a \mathbin{@} (knw_a, P) \in S}{(K_I, S) \xrightarrow{ev} (K_I', \{a \mathbin{@} (knw_a', P')\} \cup S \setminus \{a \mathbin{@} (knw_a, P)\})}$$

untappable communications. As no agent, nor the intruder, except for the sending agent and the receiving agent, are aware of untappable communications, we model them as synchronous communication. This captures both untappable send and receive in one transition. Hence, a new label for this transition is needed. We use $uc(a, b, \varphi)$, which must match both the send $us(a, b, \varphi)$ and the receive $ur(a, b, \varphi)$ events. Note that the intruder's knowledge does not change due to untappable communications, nor does the sending agent's knowledge.

$$\frac{\begin{array}{c}(K_I, knw_a, P_a) \xrightarrow{us(a,b,\varphi)} (K_I, knw_a, P_a') \\ (K_I, knw_b, P_b) \xrightarrow{ur(a,b,\varphi)} (K_I, knw_b', P_b') \\ s_0 = \{a \mathbin{@} (knw_a, P_a), b \mathbin{@} (knw_b, P_b)\} \qquad s_0 \subseteq S\end{array}}{(K_I, S) \xrightarrow{uc(a,b,\varphi)} (K_I, \{a \mathbin{@} (knw_a, P_a'), b \mathbin{@} (knw_b', P_b')\} \cup S \setminus s_0)}$$

phase synchronisation. Phase events denote synchronisation points. A $ph(i)$ event may only occur if all authorities have agreed that the election will evolve into a new phase. As a consequence, all agents who are ready and willing to do so will move to the new phase as well.[1]

In the semantics rule below, the agents that are ready and willing to execute the phase transition are captured (together with their states) in the set *Phase*. Note that *Phase* is a subset of all agents ready to execute a phase transition, as readiness does not imply willingness. The set $Phase'$ reflects the new states for these agents. Finally, we explicitly require each authority

[1] We conjecture that our semantics of phase synchronisation is similar to the strong phase semantics as proposed in [10].

$a \in \mathit{Aut}$ to be ready and willing to execute the phase transition.

$$\frac{\begin{array}{c} i \in \mathbb{N} \\ \mathit{Phase} \subseteq \{a \mathbin{@} (\mathit{knw}_a, P_a) \in S \mid \exists P'_a \colon (K_I, \mathit{knw}_a, P_a) \xrightarrow{ph(i)} (K_I, \mathit{knw}_a, P'_a)\} \\ \mathit{Aut} \subseteq \{a \in \mathit{Agents} \mid \exists \mathit{knw}_a, P_a \colon a \mathbin{@} (\mathit{knw}_a, P_a) \in \mathit{Phase}\} \\ \mathit{Phase}' = \{a \mathbin{@} (\mathit{knw}_a, P'_a) \mid \exists P_a \colon a \mathbin{@} (\mathit{knw}_a, P_a) \in \mathit{Phase} \wedge \\ \qquad (K_I, \mathit{knw}_a, P_a) \xrightarrow{ph(i)} (K_I, \mathit{knw}_a, P'_a)\} \end{array}}{(K_I, S) \xrightarrow{ph(i)} (K_I, \mathit{Phase}' \cup S \setminus \mathit{Phase})}$$

The above semantics rules give rise to labelled transition systems. Each possible execution of the system is represented by a path in this labelled transition system. A path is represented by a list of the path's labels and is called a trace. The set of traces of a given voting system is defined as follows.

Definition 9 (traces)**.** *The class of traces Traces consists of finite lists of labels. The traces of a voting system $\mathcal{VS}^\gamma$ (voting system $\mathcal{VS}$ instantiated with choice function γ) are given by*

$$\begin{array}{rcl} \mathit{Tr}(\mathcal{VS}^\gamma) & = & \{\alpha \in \mathit{Labels}^\star \mid \alpha = \alpha_0 \dots \alpha_{n-1} \ \wedge \\ & & \quad \exists s_0, \dots, s_n \in \mathit{State} \colon s_0 = (K_I^0, \mathcal{VS}^\gamma) \ \wedge \\ & & \quad \forall 0 \leq i < n \colon s_i \xrightarrow{\alpha_i} s_{i+1}\} \end{array}$$

The set of traces of a voting system $\mathcal{VS}$ is now given by

$$\mathit{Tr}(\mathcal{VS}) = \bigcup_{\gamma \in \mathcal{V} \to \mathcal{C}} \mathit{Tr}(\mathcal{VS}^\gamma).$$

We denote the intruder knowledge in the last state of a trace t as K_I^t. The empty trace is denoted by ϵ.

Example (traces). Consider the voting system $\mathcal{VS}_0$ from Table 2, and a choice function γ_1 such that $\gamma_1(v1) = \gamma_1(v2) = c$. Then we have the following.

$$\begin{array}{ll} \epsilon & \in \mathit{Tr}(\mathcal{VS}_0^{\gamma_1}) \\ s(v1, T, \{\{c\}_{sk(v1)}\}_{pk(T)}) & \in \mathit{Tr}(\mathcal{VS}_0^{\gamma_1}) \\ s(v1, T, \{\{c\}_{sk(v1)}\}_{pk(T)})\ s(v2, T, \{\{c\}_{sk(v2)}\}_{pk(T)}) & \in \mathit{Tr}(\mathcal{VS}_0^{\gamma_1}) \\ s(v1, T, \{\{c\}_{sk(v1)}\}_{pk(T)})\ r(v1, T, \{\{c\}_{sk(v1)}\}_{pk(T)}) & \in \mathit{Tr}(\mathcal{VS}_0^{\gamma_1}) \\ \dots & \end{array}$$

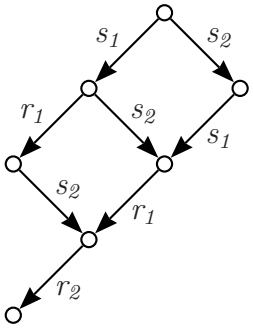

Figure 1: A labelled transition system

If we denote the send event of voter *v1* as s_1, the send event of voter *v2* as s_2 and the corresponding read events of the tallyer as r_1, r_2, respectively, the labelled transition system for this voting system is as in Figure 1. Its full set of traces is succinctly described as

$$\begin{aligned} Tr(\mathcal{VS}_0^{\gamma_1}) \;=\; & \{\epsilon, s_1, s_2, s_1 s_2, s_2 s_1, s_1 r_1, s_1 s_2 r_1, s_2 s_1 r_1, s_1 r_1 s_2, \\ & s_1 s_2 r_1 r_2, s_1 r_1 s_2 r_2, s_2 s_1 r_1 r_2\}. \end{aligned}$$

Traces model the dynamic behaviour of the system. The next section determines the privacy of a given voter in a given trace. This is then extended to establish the privacy of a voter in a voting system.

4. Privacy in voting systems

The framework developed in the previous section enables us to express if an intruder can distinguish two executions of the system, as previously expressed by Mauw, Verschuren and De Vink [18] and later by Garcia et al. [17] for passive intruders. Traces t, t' are to be considered equivalent if the intruder cannot distinguish them. To formalise this equivalence, the distinguishing ability of the intruder is formalised as the intruder's ability to distinguish two messages. We introduce the notion of *reinterpretation* to capture this.

Definition 10 (reinterpretation [17])**.** *Let ρ be a permutation on the set of terms Terms and let K_I be a knowledge set. The map ρ is a* semi-

reinterpretation under K_I *if it satisfies the following.*

$$\begin{array}{rcll} \rho(\varphi) & = & \varphi & \text{for } \varphi \in \mathcal{C} \cup \{a, sk(a), pk(a) \mid a \in Agents\} \\ \rho((\varphi_1, \varphi_2)) & = & (\rho(\varphi_1), \rho(\varphi_2)) & \\ \rho(\{\varphi\}_k) & = & \{\rho(\varphi)\}_{\rho(k)} & \text{if } K_I \vdash \varphi, k \ \vee \ K_I \vdash \{\varphi\}_k, k^{-1} \end{array}$$

Map ρ is a reinterpretation under K_I *iff it is a semi-reinterpretation and its inverse ρ^{-1} is a semi-reinterpretation under $\rho(K_I)$. The notion of reinterpretation is extended straightforwardly to events and to traces by applying ρ to the message fields of events (in traces).*

The notion of reinterpretation models the intruder's ability to distinguish terms. The intruder can distinguish any candidate and agent name from any other term. As public keys and private keys identify the agent, these too can be distinguished from any other term. As the intruder does not know which nonces or (non-public / private) keys belong to which agents, he cannot distinguish these from other terms. Furthermore, the intruder can distinguish the structure of paired terms. Encrypted terms can only be distinguished if the intruder can decrypt the term or if he can construct the term himself. Note that as the intruder cannot distinguish one key from another, ρ must be applied to the encryption key of any encrypted message that he can distinguish.

Example (reinterpretation). In Table 3, we provide two permutations on terms ρ and ρ' that are reinterpretations for $k, k', n, n', n'' \in K_I$.

φ	$\rho(\varphi)$	$\rho'(\varphi)$
n	n'	n''
k	k'	k
c	c	c
$\{(c, n)\}_k$	$\{(c, n')\}_{k'}$	$\{c, n''\}_k$
$sk(a)$	$sk(a)$	$sk(a)$

Table 3: Example: reinterpretation of terms

Some events in a trace are hidden from the intruder, hence the intruder has a restricted view of a trace. In particular, the intruder cannot see any *uc* transitions (communications over untappable channels), nor the sender of

anonymous communications. The visible part of a trace is captured by the function $obstr\colon Traces \to Traces$ as follows:

$$obstr(\epsilon) = \epsilon$$
$$obstr(\ell \cdot t) = \begin{cases} obstr(t) & \text{if } \ell = uc(a, a', \varphi) \\ as(x, \varphi) \cdot obstr(t) & \text{if } \ell = as(, x, \varphi) \\ \ell \cdot obstr(t) & \text{otherwise} \end{cases}$$

Definition 11 (trace indistinguishability)**.** *Traces t, t' are indistinguishable for the intruder, notation $t \sim t'$ iff there exists a reinterpretation ρ such that the visible part of t is the visible part of $\rho(t')$ and the final intruder knowledge in t and t' is equal modulo ρ. Formally put:*

$$t \sim t' \equiv \exists \rho\colon obstr(t') = \rho(obstr(t)) \ \wedge \ \overline{K_I^t} = \rho(\overline{K_I^{t'}}).$$

The above definition of the intruder's ability to distinguish traces extends to his ability to distinguish sets of traces as follows.

Definition 12 (choice indistinguishability)**.** *Given voting system $\mathcal{VS}$, choice functions γ_1, γ_2 are indistinguishable to the intruder, notation $\gamma_1 \simeq_{\mathcal{VS}} \gamma_2$ iff*

$$\forall t \in Tr(\mathcal{VS}^{\gamma_1})\colon \exists t' \in Tr(\mathcal{VS}^{\gamma_2})\colon t \sim t' \quad \wedge$$
$$\forall t \in Tr(\mathcal{VS}^{\gamma_2})\colon \exists t' \in Tr(\mathcal{VS}^{\gamma_1})\colon t \sim t'$$

The set of choice functions indistinguishable for the intruder in a given system is now succinctly defined as follows.

Definition 13 (choice group)**.** *The choice group for a voting system $\mathcal{VS}$ and a choice function γ is given by*

$$cg(\mathcal{VS}, \gamma) = \{\gamma' \mid \gamma \simeq_{\mathcal{VS}} \gamma'\}.$$

The choice group for a particular voter v, i.e. the set of candidates indistinguishable from v's chosen candidate, is given by

$$cg_v(\mathcal{VS}, \gamma) = \{\gamma'(v) \mid \gamma' \in cg(\mathcal{VS}, \gamma)\ \}.$$

In the last definition, the privacy of a voting system is defined with respect to an intruder who can control all communication channels except the untappable channels. The next chapter poses the question of how much of this remaining privacy is controlled by the voter.

5. Conspiring voters

The above framework captures the behaviour of a passive voter, who does not actively cooperate with the intruder to prove how she has voted. However, as remarked in the introduction, we focus on voters trying to renounce their vote-privacy. A conspiring voter can try to share her knowledge with the intruder. The classic receipt-freeness case assumes the voter shares her final knowledge. As noted in [12], the timing of knowledge sharing is important. In order to prove that she really has the receipt, the voter needs to share her private knowledge before it becomes public in the course of the execution of the voting system. Furthermore, during the course of an election, a voter may learn or commit to knowledge that the intruder is unaware of, by communicating over untappable channels. A voter may seek to circumvent the privacy provisions of these channels by sharing knowledge received over such a channel with the intruder, and by using intruder-supplied information to send over such a channel. The timing of sharing information between the conspiring voter and the intruder hence is important.

We distinguish the cases where the voter shares her full knowledge (post-election or pre-election) from the cases where the voter conspires due to untappable channels (sharing information, using intruder-supplied information, or both). In absence of untappable channels, all communications are visible to the intruder. In this case, the sooner a voter shares her knowledge with the intruder, the more traces the intruder can distinguish. Classical receipt-freeness, *classic-rf*, tries to break vote-privacy by sharing knowledge after elections. However, sharing knowledge beforehand, *start-rf*, gives the intruder more knowledge during the elections. This situation is depicted below in Figure 2(i).

In presence of untappable channels, the intruder is not aware of every increase of the voter's knowledge. The voter can mitigate this by conspiring mid-election. Her willingness to do so is captured in Figure 2(ii). The conspiring voter may choose to share information the intruder would not learn otherwise (*rf-share*) or use intruder-supplied terms in communications hidden from the intruder (*rf-witness*) to later prove how she voted. The combination of these two notions is at the top of the ordering (*rf-relay*).

A voter may use privacy-reducing techniques from both hierarchies to reduce her privacy. We denote this, e.g., as a *type 1a* voter, or a *type 2c* voter. In the next section, we present precise definitions of these notions.

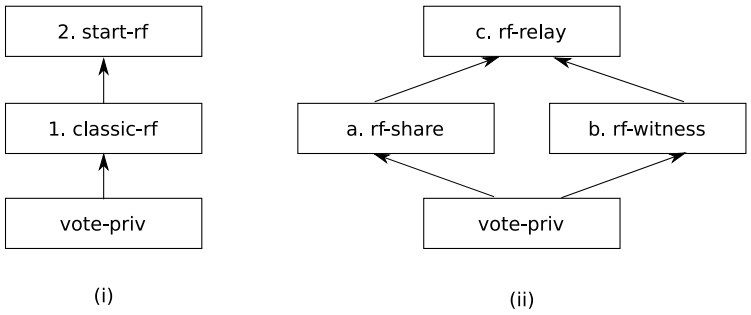

Figure 2: Knowledge sharing, (i) pre- and post-election and (ii) mid-election

6. Modelling conspiratory behaviour

A conspiring voter behaves differently from a regular voter, as she will communicate with the intruder in certain circumstances. We incorporate the different conspiracy classes into the framework as follows. We start by selecting a regular voter and the conspiracy class that we assume her to satisfy. Then the regular specification of this voter is transformed into a conspiring specification by following the transformation rules of this conspiracy class as defined below. For instance, the transformation for class 1 (classic-rf) consists of adding an extra event at the end of the voter's specification which represents the sharing of her knowledge with the intruder.

In order to formalise this approach, we extend the set of events Ev with events: $\{is(\varphi), ir(\varphi)\}$, where $is(\varphi)$ denotes the agent sending term φ to the intruder, and $ir(\varphi)$ denotes receiving term φ from the intruder. The agent level semantics of these events is given below:

send to intruder:

$$\frac{knw_v \vdash \varphi \qquad \mathrm{fv}(\varphi) = \emptyset}{(K_I, knw_v, is(\varphi).P) \xrightarrow{is(\varphi)} (K_I \cup \{\varphi\}, knw_v, P)}$$

receive from intruder:

$$\frac{\begin{array}{cc} K_I \vdash \varphi' & \mathrm{fv}(\varphi') = \emptyset \\ \mathrm{match}(\varphi', \varphi, \sigma) & \mathrm{Rd}(knw_v, \varphi) \end{array}}{(K_I, knw_v, ir(\varphi).P) \xrightarrow{ir(\varphi')} (K_I, knw_v \cup \{\varphi'\}, \sigma(P))}$$

The specific conspiracy classes are captured by a syntactical transformation of the process of the conspiring voter v as follows:

- **1. classic-rf:** at the end of her process, v sends her knowledge set to the intruder (knw_v is represented as a pairing of each element of knw_v).

- **2. start-rf:** as the first event executed by v, she sends her knowledge set to the intruder.
- **a. rf-share:** each $ur(a, v, \varphi)$ is followed by an $is(\varphi)$.
- **b. rf-witness:** The intruder supplies the voter-controllable parts of the term used in each $us(v, a, \varphi)$. To do so, the intruder must know what terms are voter-controllable. To this end, we introduce two functions:
  - a function $\text{vars}(v, \varphi)$ that returns the variables of φ that agent v can control, and
  - a function $\text{freshvars}(v, \varphi)$, that replaces every voter-controllable variable in φ by a fresh variable.

  The voter sends $\text{vars}(v, \varphi)$ to the intruder, who replies with a similar term, changing the values to his liking. The voter then uses the newly supplied values in the untappable send event.
- **c. rf-full:** this combines rf-share and rf-witness.

Note that where classic-rf and start-rf transform the ending and beginning, respectively, of voter v's process, the other three conspiracy classes transform events inside the voter process. To model conspiracy, we first introduce an event transformation function θ_i, which relies on auxiliary functions vars and freshvars. Then, the event transformation function is then extended to processes and to voting systems.

The function $\text{vars}\colon \mathcal{V} \times \mathit{Terms} \to \mathit{Terms}$ captures those variables of a term φ that are under a voter v's control. This function is defined as follows.

$$\text{vars}(v, \varphi) = \begin{cases} \{\varphi\} & \text{if} \quad \varphi \in \mathsf{Vars} \\ \text{vars}(v, \varphi') & \text{if} \quad \varphi = \{\varphi'\}_k, \text{ for } k \in \mathit{Keys}_v \\ \text{vars}(v, \varphi_1) \cup \text{vars}(v, \varphi_2) & \text{if} \quad \varphi = (\varphi_1, \varphi_2) \\ \emptyset & \text{otherwise} \end{cases}$$

The function $\text{freshvars}\colon \mathcal{V} \times \mathit{Terms} \to \mathit{Terms}$ replaces every voter-controllable variable in φ by a fresh variable. To this end, we assume the existence of a substitution σ_{fresh}, where $\text{dom}(\sigma_{\mathit{fresh}}) = \text{vars}(v, \varphi)$, that substitutes fresh variables for every voter-controlled variable. Note that some, but not all occurrences of a variable may be voter-controllable (e.g. var in the term

$(\mathsf{var}, \{\mathsf{var}\}_{sk(Reg)}))$. Hence, σ_{fresh} may only freshen those those uses of variables that are under voter control, as follows.

$$\text{freshvars}(v, \varphi) = \begin{cases} \sigma_{fresh}(\mathsf{var}) & \text{if} \quad \varphi \in \mathsf{Vars} \\ \{\text{freshvars}(v, \varphi')\}_k & \text{if} \quad \varphi = \{\varphi'\}_k, \text{ for } k \in Keys_v \\ (\text{freshvars}(v, \varphi_1), \text{freshvars}(v, \varphi_2)) & \text{if} \quad \varphi = (\varphi_1, \varphi_2) \\ \varphi & \text{otherwise} \end{cases}$$

Definition 14 (event transformation)**.** *The event transformation functions* $\theta_i \colon \mathcal{V} \times Ev \to Processes$*, for* i *a conspiracy class* $\in \{a, b, c\}$*, are defined as follows.*

- $\theta_a(v, ev) = \begin{cases} ur(ag, v, \varphi) \ . \ is(\varphi) & \text{if } ev = ur(ag, v, \varphi) \\ ev & \text{otherwise} \end{cases}$

- $\theta_b(v, ev) = \begin{cases} is(\text{vars}(v, \varphi)) \ . \ ir(\text{vars}(v, \varphi')) \ . \ us(v, ag, \varphi') \\ \qquad \text{if } ev = us(v, ag, \varphi), \text{ for } \varphi' = \text{freshvars}(v, \varphi) \\ ev \qquad \text{otherwise} \end{cases}$

- $\theta_c(v, ev) = \theta_b(v, \theta_a(v, ev))$.

We model sending of a set (for example, $is(\text{vars}(v, \varphi))$*) as sending a pairing of all elements of the set (in this example,* $\text{vars}(v, \varphi)$*).*

The event transformation θ_a ensures that every untappable receive event is followed by an intruder send event. The event transformation θ_b ensures that the intruder supplies all voter-controllable input for every untappable send event. The event transformation θ_c combines θ_a and θ_b.

Example (event transformation). In Table 4, we show some examples of event rewrites. We leave out θ_c, as it is the composition of θ_a and θ_b.

We model a conspiring voter, according to the conspiracy classification above, by means of a process transformation function. This function transforms the process by introducing conspiring behaviour.

Definition 15 (process transformation)**.** *The process transformation function* $\Theta_i \colon \mathcal{V} \times Processes \to Processes$ *transforms a process for a specific voter* v

ev	$\theta_a(ev)$	$\theta_b(ev)$
$ph(1)$	$ph(1)$	$ph(1)$
$ur(a, b, \mathsf{var})$	$ur(a, b, \mathsf{var})\ .\ is(\mathsf{var})$	$ur(a, b, \mathsf{var})$
$us(a, b, (\mathsf{vc}, \{\mathsf{vc}\}_{sk(Reg)}))$	$us(a, b, (\mathsf{vc}, \{\mathsf{vc}\}_{sk(Reg)}))$	$is(\mathsf{vc})\ .\ ir(\mathsf{irvar})\ .$ $us(a, b, (\mathsf{irvar}, \{\mathsf{vc}\}_{sk(Reg)}))$

Table 4: Example: event transformation

into a conspiring process of the class $i \in \{1, 2, a, b, c\}$*. For* $i = 2$*, conspiracy is a matter of sharing initial knowledge, which is modelled as*

$$\Theta_2(v, P) = is(knw_v).P$$

In the other cases, conspiracy has an effect on the events of the processes. For readability, we do not distinguish cases for $i = 1$*, but simply state* $\theta_1(v, ev) = ev$*. For* $i \in \{1, a, b, c\}$*, process transformation is defined as follows.*

$$\Theta_i(v, P) = \begin{cases} \delta & \text{if } \ i \neq 1 \wedge P = \delta \\ is(knw_v).\delta & \text{if } \ i = 1 \wedge P = \delta \\ \theta_i(v, ev).\Theta_i(v, P) & \text{if } \ P = ev.P \\ \Theta_i(v, P_1) + \Theta_i(v, P_2) & \text{if } \ P = P_1 + P_2 \\ \Theta_i(v, P_1) \lhd \varphi_1 = \varphi_2 \rhd \Theta_i(v, P_2) & \text{if } \ P = P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2, \\ & \quad \text{for } \varphi_1, \varphi_2 \in \mathit{Terms} \\ \theta_i(v, ev).Y(\varphi_1, \ldots, \varphi_n), & \text{for fresh } Y(\mathsf{var}_1, \ldots, \mathsf{var}_n) = \Theta_i(v, P') \\ & \text{if } P = X(\varphi_1, \ldots, \varphi_n) \wedge X(\mathsf{var}_1, \ldots, \mathsf{var}_n) = P' \end{cases}$$

Example (process transformation). In the example process transformations below, s_1 is the send action of voting system $\mathcal{VS}_0$, in Table 2. The first two entries transform voter process 1 from $\mathcal{VS}_0$. The final example illustrates how a more complex process is transformed.

P	i	$\Theta_i(v, P)$
$s_1.\delta$	1	$s_1.is(\ (pk(T), sk(v1))\ ).\delta$
$s_1.\delta$	2	$is(\ (pk(T), sk(v1))\ ).s_1.\delta$
$s_1.\delta + s_2.\delta$	$\in \{a, b, c\}$	$\theta_i(v1, s_1).\delta\ +\ \theta_i(v1, s_2).\delta$

Process transformation is extended to voting systems as follows.

$$\Theta_i(v, \mathcal{VS})(a) = \begin{cases} \mathcal{VS}(a) & \text{if } a \neq v \\ (\pi_1(\mathcal{VS}(v)), \Theta_i(v, \pi_2(\mathcal{VS}(v)))) & \text{if } a = v \end{cases}$$

The above transformations are extended to $i \in \{1a, 2a, 1b, 2b, 1c, 2c\}$ for combinations of conspiring behaviour, e.g. $\Theta_{1a}(v, \mathcal{VS}) = \Theta_1(v, \Theta_a(v, \mathcal{VS}))$. Using the above system transformation, we can define the choice group for conspiring voters in a voting system within our framework (see Section 4).

Definition 16 (conspiracy induced choice group)**.** *The choice group of conspiring voter v in voting system $\mathcal{VS}$ given choice function γ, with respect to different conspiracy classes $i \in \{1, 2, a, b, c, 1a, 2a, 1b, 2b, 1c, 2c\}$, is given by*

$$cg_v^i(\mathcal{VS}, \gamma) = cg_v(\Theta_i(v, \mathcal{VS}), \gamma).$$

Given this definition of privacy, conspiracy-resistance is a measure of the voter's choice groups as follows.

Definition 17 (conspiracy-resistance)**.** *We call voting system $\mathcal{VS}$* conspiracy-resistant *for conspiring behaviour $i \in \{1, 2, a, b, c\}$ iff*

$$\forall v \in \mathcal{V}, \gamma \in \mathcal{V} \to \mathcal{C} \colon cg_v^i(\mathcal{VS}, \gamma) = cg_v(\mathcal{VS}, \gamma).$$

Remark that for when $|\mathcal{V}| = 1$ or $|\mathcal{C}| = 1$, we have $\forall \gamma \colon cg(\mathcal{VS}, \gamma) = \{\gamma\}$, which implies that $\forall v \in \mathcal{V} \colon cg_v(\mathcal{VS}, \gamma) = cg_v^i(\mathcal{VS}, \gamma) = \{\gamma(v)\}$. Thus, in such settings, there is not vote-privacy to lose, and conspiracy-resistance is satisfied trivially.

The notion of receipt-freeness as introduced by Benaloh and Tuinstra [1] coincides with $\forall v, \gamma \colon |cg_v^1(\mathcal{VS}, \gamma)| > 1$. The approach of Delaune, Kremer and Ryan [8, 9] coincides with choice groups of size greater than one. Our framework captures any modifier of privacy, including modifiers that are not privacy-nullifying. The amount of conspiracy-resistance of a system is measured as the difference in privacy between a regular voter and a conspiring voter. The above privacy definitions capture this by determining the exact choice group and thus the exact voter privacy for any level of conspiracy.

7. Examples

The above presented framework is designed to provide a formal and quantitative analysis. However, a full analysis of an existing voting system goes beyond the scope of this paper. Therefore, this section only illustrates application of the framework at a high level. For this purpose, models of the FOO [14] system and the ThreeBallot system (3BS, [23]) are examined.

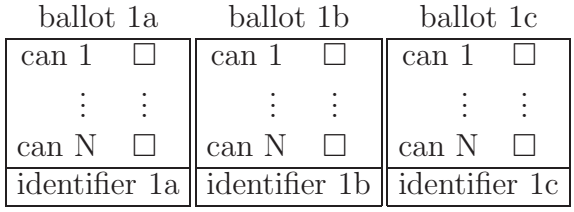

Figure 3: A Threeballot in 3BS.

7.1. FOO

The FOO protocol has been studied in literature and is known to have receipts [12, 13]. FOO does not use untappable channels, thus there are no conspiring voters of type a, b, c. The only possibility for a voter to conspire is by sharing knowledge as in Figure 2(i). A conspiring voter of type 1 already nullifies her privacy – $\forall v, \gamma\colon |cg_v^1(FOO, \gamma)| = 1$. This is because the keys used to encrypt her preferred candidate are shared with the intruder. Thus, the intruder can open these encryptions and can see how the voter voted. As such, any reinterpretation of the voter's message carrying her vote is impossible.

7.2. ThreeBallot

In 3BS, a vote is split over three single ballots (see Figure 3), which together form one Threeballot. Each ballot carries a unique identifier. Upon voting, the three ballots are cast and the voter takes home a receipt (certified copy) of one ballot. The copy allows the voter to verify that her Threeballot is actually cast. To vote for a candidate, a voter ticks two boxes in the row of that candidate; every other candidate-row only receives one tick mark. The voter is free to place the ticks in any column, as long as there is one row with two ticked boxes (her choice) and all other rows have one ticked box. Given the specific way of voting in 3BS, only a limited subset of the cast ballots can form a valid Threeballot with a given receipt (to be more precise, only those ballots combined with the receipt such that there is only one row with two tick marks). For example, consider a receipt with a tick mark for every candidate. This can only be matched with one entirely blank ballot, and one ballot containing precisely one tick mark.

An obvious attack (already pointed out by the designer in [24]) is to agree a priori with the intruder on how to fill in the ballots (captured by class b conspiracy). The intruder can then easily verify if all three ballots are cast.

This reduces privacy more strongly than a voter merely showing her receipt after the elections (class 1 conspiracy). This, in turn, gives less privacy than not showing the receipt: $cg_v^b(3BS, \gamma) \subseteq cg_v^1(3BS, \gamma) \subseteq cg_v(3BS, \gamma)$.

As pointed out in [23], 3BS can also be used in elections where voters are allowed to vote for multiple candidates. In this case, a Threeballot may contain multiple rows with two tick marks. This means that the number of ballots forming a valid Threeballot with a given receipt is increased. As the number of valid combinations directly affects privacy, voter privacy is improved. In the framework, this improvement is precisely captured by the size of choice groups.

8. Discussion

In the previous section, we have shown that our framework is able to quantify privacy loss in 3BS, where the receipt of a voter reduces the size of her choice group, but not necessarily to one. This kind of attack cannot be analysed by other formal approaches as discussed in Section 1. Moreover, our framework can capture previously established privacy nullification attacks as well as a new class of privacy reduction attacks.

An example of such new attacks is the *not-voted-for* attack, where the intruder does not want the voter to vote for a specific candidate, say $c1$. In this case, the voter's privacy need not be nullified. This is conspiracy-dependent if the intruder can rule out the possibility of the voter having voted $c1$ only because of voter conspiracy. In terms of the framework, the intruder gains sufficient information to coerce voter v if $\forall\gamma\colon c1 \notin cg_v(\Theta_{2c}(v, \mathcal{VS}), \gamma)$. Furthermore, the attack relies on voter v's cooperation if (the intruder gains enough information and) $\forall\gamma\colon c1 \in cg_v(\mathcal{VS}, \gamma)$.

A similar new type of attack manifests itself in Dutch national elections. In these elections, the candidates provided by the system are individuals, and every candidate is affiliated with precisely one party. Although voters cannot vote for parties directly, the common view of the election is that of voters voting for parties. In this setting, privacy of a voter should not merely encompass the chosen candidate, but also the affiliated party. Thus, even if $\forall\gamma\colon |cg_v(\Theta_{2c}(v, \mathcal{VS}), \gamma)| > 1$ for a specific voter v, the intruder may still be able to determine for which party v voted, and thus coerce the voter.

Another example is related to *coalitions* between the intruder and several successfully conspiring voters. Based on the final result of the election and the known votes of these voters, the intruder can further reduce the privacy of

honest voters. A simple example of such a scenario is when a conspiring voter cast the only vote for a particular candidate, then the intruder additionally learns that no other voter voted for that candidate.

Finally, the result of an election provides a bias for a voter's choice – the probability of a voter having voted for the winner is higher than the probability of a voter having voted for a losing candidate. Our framework makes it possible to account for such information bias.

The rest of this section discusses how the privacy definitions of the framework relate to the notions of receipt-freeness and coercion-resistance.

8.1. Receipt-freeness

Receipt-freeness, as introduced by Benaloh and Tuinstra [1], captures ways for a voter to reduce the privacy of how she voted. The framework captures receipt-freeness by specifying the desired class of conspiring behaviour. From the hierarchy in Figure 2, we conclude that conspiring behaviour of type $2c$ models the most conspiring behaviour in the system. Any knowledge possessed by the voter is shared as soon as possible with the intruder. Nevertheless, this does not preclude $2c$-conspiracy-resistant voting systems. For example, designated verifier proofs still force privacy on the voter, as the intruder has no way to determine if a designated verifier proof forwarded by the voter is fake or not.

8.2. Coercion-resistance

Coercion-resistance, as introduced by Juels, Catalano and Jakobsson [2], captures ways for a voter to reduce the privacy of how she interacts with the voting system. There is confusion in literature about the difference between the notions of coercion-resistance and receipt-freeness, and with good cause: if the voter can prove how she voted, the coercer can force her to produce this proof. Conversely, if there is no such proof, the coercer cannot force this proof. Thus, the susceptibility of a voter to be forced to vote in a specific way is equivalent to her ability to prove that she voted in that specific way. Succinctly put:

Proving ability = coercion susceptibility.

The notion of coercion-resistance as introduced by Juels et al. extends beyond reducing privacy of the vote. In their view, coercion-resistance encompasses the following:

- **receipt-freeness:** the voter proves how she voted to the intruder.
- **forced abstention:** the intruder prevents the voter from voting.
- **simulation attacks:** the voter gives her private keys to the intruder, who votes in her stead.
- **forced random voting:** the intruder forces the voter to vote for a random entry in a list of encrypted candidates. Note that the intruder does not need to know which candidate is chosen, as long as it is a random choice. This attack (if applied to many voters) forces a more uniform distribution of votes instead of the actual distribution of votes. Such a strategy benefits candidates with less than average votes at the cost of harming candidates with more than average votes.

Receipt-freeness is captured by the framework, as explained above. To capture forced abstention, we extend the range of γ with an "abstention" candidate $\perp$. This candidate embodies all votes, that do not affect the result (we do not distinguish between abstention and malformed voting). The extended set is referred to as $\mathcal{C}_\perp$. Using these extensions, the intruder can force voter v with conspiring behaviour i to abstain, iff

$$cg_v^i(\mathcal{VS}, \gamma) = \{\perp\}.$$

In a simulation attack, the intruder casts a vote himself. Simulation attacks are resisted if the intruder cannot tell if the vote he cast affects the result or not. While the type of voter behaviour necessary for a simulation attack is modelled by the framework (type 2 behaviour), intruder voting is not. However, if we extend the domain of γ to include the intruder int. The extended set is referred to as $\mathcal{V}_{int}$. The choice group of the intruder then captures the intruder's unsureness about his own vote. Hence, a system is simulation-resistant if the intruder cannot tell whether his vote is counted or not, i.e.

$$\{\perp, \gamma(int)\} \subseteq cg_{int}(\mathcal{VS}, \gamma).$$

In forced random voting attacks, the intruder forces the voter to vote randomly. This means that whenever the voter can make a choice, either conditionally or non-deterministically, the intruder forces his choice on the voter. This can be expressed in terms of the framework by rewriting every process P that denotes a choice. Let Δ_{rnd} be a process transformation

function for forcing choices. Then we have, for any process P such that $P = P_1 + P_2$ and for any process P such that $P = P_1 \lhd \varphi_1 = \varphi_2 \rhd P_2$,

$$\Delta_{rnd}(P) = ir(\mathsf{var}).\Delta_{rnd}(P_1) \ \lhd \mathsf{var} = \mathit{true} \rhd \ \Delta_{rnd}(P_2),$$

where var is a fresh variable and *true* is a constant term $\in \mathit{Terms}$.

Thus, coercion attacks can be captured in the framework as well. However, we keep these latter attacks separate, as these attacks are fundamentally different from the privacy-reducing conspiratory model.

9. Conclusions

We have developed a framework to precisely measure voter-controlled privacy, with respect to different ways how one voter can share her knowledge with the intruder. This quantitative framework is based on knowledge reasoning and trace equivalences. It allows us to capture the exact meaning of receipt-freeness in the context of vote buying, and to detect attacks that have escaped the focus of published methods in the literature as well.

We intend to apply the framework in full detail to several voting systems. Our quantitative approach to privacy in voting will, we believe, uncover previously unidentified attacks. We are also interested in modelling authorities conspiring with the intruder (an extension from conspiring voters), and in investigating the potential effects of various counting methods on privacy loss.

References

[1] J. Benaloh, D. Tuinstra, Receipt-free secret ballot elections (extended abstract), in: Proc. 26th ACM Symposium on the Theory of Computing, ACM, 1994, pp. 544–553.

[2] A. Juels, D. Catalano, M. Jakobsson, Coercion-resistant electronic elections, in: Proc. 4th ACM Workshop on Privacy in the Electronic Society, ACM, 2005, pp. 61–70.

[3] T. Okamoto, An electronic voting scheme, in: Proc. 14th IFIP World Computer Congress, Conference on IT Tools, 1996, pp. 21–30.

[4] B. Lee, K. Kim, Receipt-free electronic voting through collaboration of voter and honest verifier, in: Proc. Japan-Korea Joint Workshop on Information Security and Cryptology, 2000, pp. 101–108.

[5] M. Hirt, K. Sako, Efficient receipt-free voting based on homomorphic encryption, in: Proc. 19th Conference on the Theory and Application of Cryptographic Techniques, Vol. 1807 of LNCS, Springer, 2000, pp. 539–556.

[6] M. Hirt, Multi-party computation: Efficient protocols, general adversaries, and voting, Ph.D. thesis, ETH Zurich (2001).

[7] B. Lee, K. Kim, Receipt-free electronic voting with a tamper-resistant randomizer, in: Proc. 4th Conference on Information and Communications Security, Vol. 2513 of LNCS, Springer, 2002, pp. 389–406.

[8] S. Delaune, S. Kremer, M. Ryan, Coercion-resistance and receipt-freeness in electronic voting, in: Proc. 19th IEEE Computer Security Foundations Workshop, IEEE Computer Society, 2006, pp. 28–42.

[9] S. Delaune, S. Kremer, M. Ryan, Verifying privacy-type properties of electronic voting protocols, Journal of Computer Security. To appear.

[10] S. Delaune, M. Ryan, B. Smyth, Automatic verification of privacy properties in the applied pi-calculus, in: Proc. 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security, Vol. 263 of IFIP Conference Proceedings, Springer, 2008, pp. 263–278.

[11] M. Backes, C. Hriţcu, M. Maffei, Automated verification of remote electronic voting protocols in the applied pi-calculus, in: Proc. 21st IEEE Computer Security Foundations Symposium, IEEE Computer Society, 2008, pp. 195–209.

[12] H. Jonker, E. de Vink, Formalising receipt-freeness, in: Proc. 9th Conference on Information Security, Vol. 3444 of LNCS, Springer, 2006, pp. 476–488.

[13] A. Baskar, R. Ramanujam, S. Suresh, Knowledge-based modelling of voting protocols, in: Proc. 11th Conference on Theoretical Aspects of Rationality and Knowledge, ACM, 2007, pp. 62–71.

[14] A. Fujioka, T. Okamoto, K. Ohta, A practical secret voting scheme for large scale elections, in: Proc. 3rd Workshop on the Theory and Application of Cryptographic Techniques, Vol. 718 of LNCS, Springer, 1992, pp. 244–251.

[15] T. Okamoto, Receipt-free electronic voting schemes for large scale elections, in: Proc. 5th Workshop on Security Protocols, Vol. 1361 of LNCS, Springer, 1997, pp. 25–35.

[16] R. Aditya, B. Lee, C. Boyd, E. Dawson, An efficient mixnet-based voting scheme providing receipt-freeness, in: Proc. 1st Conference on Trust and Privacy in Digital Business, Vol. 3184 of LNCS, Springer, 2004, pp. 152–161.

[17] F. Garcia, I. Hasuo, W. Pieters, P. van Rossum, Provable anonymity, in: Proc. 3rd ACM Workshop on Formal Methods in Security Engineering, ACM, 2005, pp. 63–72.

[18] S. Mauw, J. Verschuren, E. de Vink, A formalization of anonymity and onion routing, in: Proc. 9th European Symposium on Research Computer Security, Vol. 3193 of LNCS, Springer, 2004, pp. 109–124.

[19] T. Chothia, S. Orzan, J. Pang, M. Torabi Dashti, A framework for automatically checking anonymity with μCRL, in: Proc. 2nd Symposium on Trustworthy Global Computing, Vol. 4661 of LNCS, Springer, 2007, pp. 301–318.

[20] D. Dolev, A. Yao, On the security of public key protocols, IEEE Transactions on Information Theory 29 (12) (1983) 198–208.

[21] L. Aceto, W. Fokkink, C. Verhoef, Structural operational semantics, in: Handbook of Process Algebra, Elsevier, 2001, pp. 197–292.

[22] C. Cremers, Scyther - semantics and verification of security protocols, Ph.D. thesis, Eindhoven University of Technology (2006).

[23] R. Rivest, W. Smith., Three voting protocols: Threeballot, VAV, and Twin, in: Proc. 2007 USENIX/ACCURATE Electronic Voting Technology Workshop, USENIX, 2007.

[24] R. Rivest, The threeballot voting system, unpublished manuscript (2006).