BRICS

Basic Research in Computer Science

BRICS RS-01-52 J. Groth: Extracting Witnesses from Proofs of Knowledge

Extracting Witnesses from Proofs of Knowledge in the Random Oracle Model

Jens Groth

BRICS Report Series RS-01-52

ISSN 0909-0878 December 2001

Extracting Witnesses from Proofs of Knowledge in the Random Oracle Model

Jens Groth

Cryptomathic* and BRICS, Aarhus University†

Abstract

We prove that a 3-move interactive proof system with the special soundness property made non-interactive by applying the Fiat-Shamir heuristic is almost a non-interactive proof of knowledge in the random oracle model. In an application of the result we demonstrate that the Damgård-Jurik voting scheme based on homomorphic threshold encryption is secure against a nonadaptive adversary according to Canetti's definition of multi-party computation security.

1 Introduction

We study security proofs for multi-party cryptographic protocols in the random oracle model. In particular we look at the voting scheme of Damgård and Jurik [DJ01].

The random oracle model usually comes into protocol design when a hash-function is used. This can lead to very efficient protocols. However, to prove security one models the hash function as a random oracle. Such a proof is not a real proof of security, but it is still better than no proof at all.

There are general multi-party computation protocols not using random oracles [GMW87, CDN01]. For more specialized purposes such as threshold signatures and cryptosystems random oracles have been used though, see for instance [Sho00]. However, rigorous security proofs for voting schemes and multi-party computation in general in the random oracle model seem to be missing.

A standard way of using a random oracle is to turn a 3-move interactive proof into a non-interactive proof through the Fiat-Shamir heuristic. Such

*Cryptomathic A/S (`www.cryptomathic.com`).

†Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

a proof can be used for proving membership of a language. Often we need something stronger though, namely, we need to prove knowledge of a witness for membership of a language. It is the latter case we concern ourselves with in this paper.

Let us describe the setting in more detail. Assume we have a language L_R defined by some binary relation R, meaning that an element x belongs to L_R if and only if there is a witness w such that $(x, w) \in R$. In a 3-move protocol the prover sends an initial message a, the verifier responds with a randomly chosen challenge e, and the prover answers the challenge with a string z. On basis of the conversation (x, a, e, z) the verifier decides whether to accept or reject the proof of $x \in L_R$. In the random oracle model such a proof can be made non-interactive by using the random oracle answer to the query (x, a) as the challenge e.

Most examples of 3-move proof protocols have a property known as special soundness, i.e. from two different accepting conversations on the same initial message (a, e, z) and (a, e', z') it is possible to extract a witness w for $x \in L_R$. By rewinding a prover to his state just after sending out a and replying with different challenges it can be possible to gather the witness from the prover's outputs. This means that it is a proof of knowledge.

The problem is that this rewinding technique does not work in the random oracle model. We could of course on one proof (x, a, e, z) try to rewind the prover to the point where it queried (x, a), and give a different answer e' to the query hoping for a new proof (x, a, e', z') to be returned. However, this may never happen! There is no guarantee that the prover will return a proof with the same x and a.

We present a solution to this problem. We demonstrate that by controlling the prover we can with high probability extract witnesses. More precisely we show that it is possible to simulate any adversary making non-interactive proofs from a 3-move special soundness proof protocol such that the output has the same distribution, and additionally with high probability to supply witnesses for the proofs in the output.

We note that while we do get arbitrarily high probability of extracting witnesses, we do not achieve overwhelming probability of extracting witnesses. This stems from the fact that the simulator depends on the extraction probability we wish for. However, it appears that this weaker notion of a proof of knowledge is sufficient in many cases!

We use our result to prove the security of the Damgård-Jurik voting scheme according to Canetti's definition of secure multi-party computation [Can00]. To the best of our knowledge this is the first time a voting scheme has been proven secure in this sense, and it implies almost directly that the scheme satisfy standard properties required of voting schemes, such as robustness and privacy for instance. We believe the proof technique is generally applicable in security proofs of protocols. In particular we note that security of the El-Gamal based voting scheme in [CGS97] can be shown in a quite similar way as we show the security of the Damgård-Jurik scheme.

Pointcheval and Stern [PS00] have looked at a related question in the context of signature schemes in the random oracle model. These schemes are also based on 3-move protocols as described above. The public key contains an $x \in L_R$, the private key is w such that $(x, w) \in R$, and a signature can be seen as a proof of knowledge of w. One can prove such a scheme secure by proving that forging a signature efficiently implies knowledge of w. This cannot happen with significant probability if we assume w is infeasible to compute from x. Thus this is similar to, but not the same, as the question we look at here. One difference is that in our case, the adversary is free to choose the instance x he proves knowledge for, while an adversary against the signature scheme has a harder time since he must work with the x specified in the public key.

Finally we remark that the setting we work in is a bit more general than standard 3-move proofs. We consider what we call proofs relative to a language L. The proof system for L_R works as a normal proof system if x also belongs to some other language L, while there are no guarantees if $x \notin L$. Setting L to be the language of all finite strings returns us to the normal case where we simply talk of a proof system for L_R.

2 Preliminaries

We use a random oracle to model the behaviour of suitable hash-functions. Such an oracle works like this: If the query has not been made before it returns a random k-bit string. On the other hand, if the query has been made before it simply returns the same string as it at the last identical query. Obviously, a real life hash-function has a stronger relation between input and output but the model captures certain wanted features of the hash-function such as one-wayness and collision-freeness. As a heuristic method we can therefore prove a protocol secure in the random oracle model and through this get some indication that the protocol is secure also when using a real hash-function instead of a random oracle. If an algorithm A has access to an oracle $\mathcal{O}$ we write $A^{\mathcal{O}}$ to indicate this.

All algorithms have 1^k as input where k is the security parameter. We usually do not write this explicitly. By saying that algorithms run in polynomial time we mean that there is some polynomial in k bounding the time they run before terminating. We denote the set of probabilistic polynomial time algorithms by PPT. For a particular polynomial time algorithm A we let t_A be a positive integer such that k^{t_A} bounds the time used by A.

In this paper we work with multi-party protocols. We think of the parties as being interactive Turing machines. To bound the time they use we consider each of their invocations as an invocation of a PPT algorithm. This class of probabilistic polynomial time interactive Turing machines we label ITM.

We use standard notation for experiments and probabilities of certain outcomes from the experiments. Typically we give some additional inputs to the algorithms involved in a particular experiment. These inputs are intended to express the non-uniformity of the algorithms involved. By proper modifications one can remove this non-uniformity. All results also hold in a uniform setting. The additional inputs to the algorithms are always bounded by a polynomial in the security parameter.

We shall work often with elements in the multiplicative group of integers modulo some positive integer n. We assume that the elements of Z_n^* are represented in some suitable way. When putting emphasis on using an integer a in $\{0, \dots, n-1\}$ to represent an element $b \in Z_n^*$ we write $a = b \bmod n^{s+1}$.

3 Proof systems and the Fiat-Shamir heuristic

3.1 3-move interactive proof systems for a language in another language

An interactive proof system for a language L_R is a protocol between two parties, which we call respectively the prover and the verifier. The goal of the prover is to convince the verifier that some string x, bounded by a suitable polynomial, known to both parties represents an element in L_R. Definitions of interactive proof systems often use an unlimited prover, however, in this article we shall always work with parties in the protocols running in polynomial time.

Let the language L_R be defined by some relation R such that $x \in L_R$ if and only if $\exists w : R(x, w)$. If the relation R can be computed by some polynomial time algorithm then this is a standard definition of an NP-language. Of course, there may be several different relations defining the same language.

The minimum requirement of an interactive proof system for a language L_R is that it has both the completeness property and the soundness property. Completeness says that if the prover and verifier both follow the protocol correctly and the prover knows some witness for $x \in L_R$ then the prover succeeds in convincing the verifier that indeed $x \in L_R$, i.e. make the verifier end the protocol by signalling that he accepts the proof. Soundness on the other hand says that when $x \notin L_R$ then no matter how the prover behaves he is not able to convince the verifier that $x \in L_R$. By themselves these requirements are not that interesting, for NP-languages the prover can just send the witness to the verifier, but when making further demands on the proof system, such as it being zero-knowledge, things start to get more complicated.

Our goal here is to generalize the notion of interactive proof systems for a language L_R to that of interactive proof systems for a language L_R relative to a language L. Since we just need 3-move interactive proofs in this paper we shall limit the definitions to that particular case here. A protocol execution looks like this: the prover sends an initial message a, the verifier responds with a challenge e, and upon receiving an answer z from the prover the verifier outputs 1 if accepting and otherwise 0.

We define a quadruple of PPT-algorithms (P_1, P_2, V_1, V_2) to be an interactive prof system for L_R relative to L if the following requirements are satisfied:

Completeness

$$\begin{aligned}&\forall\delta > 0 \exists K \forall k > K \forall x, w : x \in L \wedge R(x, w) \Rightarrow\\&P((a, s) \leftarrow P_1(x, w); (e, t) \leftarrow V_1(x, a); z \leftarrow P_2(e, s) :\\&\quad V_2(x, i, e, z, t) = 1) > 1 - \frac{1}{k^\delta}\end{aligned}$$

Soundness

$$\begin{aligned}&\forall A_1, A_2 \in PPT \forall\delta > 0 \exists K \forall k > K \forall x, u : x \in L \wedge x \notin L_R \Rightarrow\\&P((a, s) \leftarrow A_1(x, u); (e, t) \leftarrow V_1(x, a); z \leftarrow A_2(e, s) :\\&\quad V_2(x, a, e, z, t) = 1) < \frac{1}{k^\delta}\end{aligned}$$

We think of s and t as containing the states of respectively the prover and the verifier during the execution. Without loss of generality, we shall imagine that they contain the entire history of the execution of the party, including used random bits.

We shall be interested in two additional properties of an interactive proof system for a language relative to another language. One of them, special soundness, captures the fact that if the prover can make acceptable answers to two different challenges then we get hold of a witness. This can be thought of as a strong version of an interactive proof of knowledge system since if the prover's answers yield a witness then the prover is in a sense in possession of a witness. The witness can be extracted from the two proofs using an extraction algorithm E.

Special Soundness

$$\begin{aligned}&\exists E \in PPT \forall\delta > 0 \exists K \forall k > K \forall x, w : x \in L \wedge R(x, w) \Rightarrow\\&P((a, s) \leftarrow P_1(x, w); (e, t), (e', t') \leftarrow V_1(x, a); z \leftarrow P_2(e, s);\\&\quad z' \leftarrow P_2(e', s) : R(x, E(x, a, e, e', z, z', t, t'))) > 1 - \frac{1}{k^\delta}\end{aligned}$$

The other property we shall be interested in, is honest verifier zero-knowledge. If the verifier behaves according to the protocol, he shall not be able to gain any extra knowledge by interacting with the prover. This can be captured by saying that there is a simulator that outputs a string (a, e, z, t) with a distribution indistinguishable from that of the initial message, challenge and answer produced in a real interaction between an honest prover and verifier.

Honest Verifier Zero-Knowledge

$$\begin{aligned}
&\exists S \in PPT \forall D \forall \delta > 0 \exists K \forall k > K \forall x, w, u : x \in L \wedge R(x, w) = 1 \Rightarrow \\
&P((a, s) \leftarrow P_1(x, w); (e, t) \leftarrow V_1(x, a); z \leftarrow P_2(e, s) : D(x, a, e, z, t, u) = 1) \\
&-P((a, e, z, t) \leftarrow S(x) : D(x, a, e, z, t, u) = 1) < \frac{1}{k^\delta}
\end{aligned}$$

3.2 Using a random oracle to make a proof non-interactive

Having made these general definitions we restrict ourselves in the remains of this paper to the case where the verifier picks the challenge completely at random. Under this restriction the state information t does not hold any useful knowledge for the verifier and we can therefore simply let it be the empty string in the above definitions. All proof systems we know of have this property.

From the definitions above it is not clear what powers an adversary has when given an element $x \in L \cap L_R$. We want to avoid a situation where the verifier on a particular proof can respond in two ways, i.e. has a significant probability of both accepting and rejecting. For the remains of this paper we therefore only consider proof systems where the verifier on any element and proof either accepts with overwhelming probability, or rejects with overwhelming probability. Most known examples of proof systems have this property, and in particular it is the case for those proof systems where the second part of the verifier is deterministic.

In the random oracle model it is now possible to transform a 3-move interactive proof system for language L_R relative to L, which satisfies the special honest verifier zero-knowledge property into a non-interactive zero-knowledge proof system for language L_R relative to L. The trick is simply to replace the verifiers challenge with the random oracle value taken on the initial message produced by the prover, or more precisely the oracle value on the initial message a, the string in question x and possibly some auxiliary information such as an ID of the prover. The verifier's role is now reduced to check that indeed the challenge is correctly made according to the oracle and whether the resulting proof is acceptable. The resulting proof system has the following three properties:

Completeness

$$\begin{aligned}
&\forall A \in PPT \forall \delta > 0 \exists K \forall k > K \forall u : \\
&P((x, w) \leftarrow A^{\mathcal{O}}(u); (a, e, z) \leftarrow P^{\mathcal{O}}(x, w) : \\
&\qquad x \in L \wedge R(x, w) \Rightarrow V^{\mathcal{O}}(x, a, e, z) = 1) > 1 - \frac{1}{k^\delta}
\end{aligned}$$

Soundness

$$\begin{aligned}&\forall A \in PPT \forall \delta > 0 \exists K \forall k > K \forall u : \\ &P((x,a,e,z) \leftarrow A^{\mathcal{O}}(u) : x \in L \setminus L_R \wedge V^{\mathcal{O}}(x,a,e,z) = 1) < \frac{1}{k^{\delta}}\end{aligned}$$

Zero-knowledge

$$\begin{aligned}&\exists S \in PPT \forall A, D \in PPT \forall \delta > 0 \exists K \forall k > K \forall u : \\ &P((x,w,v) \leftarrow A^{\mathcal{O}}(u); (a,e,z) \leftarrow P^{\mathcal{O}}(x,w) : \\ &\qquad x \in L \wedge R(x,w) \Rightarrow D^{\mathcal{O}}(x,a,e,z,v) = 1) \\ &-P((x,w,v) \leftarrow A^{\mathcal{O}}(u); (a,e,z) \leftarrow S(x) : \\ &\qquad x \in L \wedge R(x,w) \Rightarrow D^{\mathcal{O}'}(x,a,e,z,v) = 1) < \frac{1}{k^{\delta}}\end{aligned}$$

where $\mathcal{O}'$ is the oracle $\mathcal{O}$ modified such that on the query (x, a, aux), which is used by the verifier to check that the challenge has been correctly formed, it returns e as the answer. We assume furthermore in the zero-knowledge definition that the space of possible initial messages is sufficiently large such that the A algorithm only has negligible probability of having queried the same initial message a as the one produced by P.

Note that the element-witness pairs (x, w) are created by an algorithm A with access to the random oracle. This differs from the standard interactive definition in which one simply quantifies over all element-witness pairs instead of just a single string u. The reason for this choice is that we want to capture the relation the element-witness pair may have with the random oracle in question.[1]

The method described above is often used in connection with the Fiat-Shamir heuristic in which one replaces the challenge in an interactive proof with a hash value of the initial message and the common element in question. Assuming that the hash function works like a random oracle one gets this kind of non-interactive proof system.

3.3 A useful lemma

Imagine we have an interactive proof system for language L_R relative to L, which uses random challenges of suitable length, and which has the special soundness property. Suppose furthermore that for each $x \in L$ there can be at most one witness w such that $R(x, w)$. In that case the resulting non-interactive proof system we get in the random oracle model satisfies Lemma 1 stated below.

The idea in the lemma is the following. We consider an algorithm producing some elements in L and some relative proofs that $x \in L_R$. We find

[1]Note also that a simple deletion of u transforms the formulas into a uniform setting.

that one could substitute this algorithm with a simulator that produces the same kind of output as the original algorithm, but in addition provides the witnesses for the elements belonging to L_R. By choosing a proper simulator we can make any distinguisher's chance of differentiating between the two scenarios arbitrarily small. If we think of the particular algorithm as part of a larger protocol the lemma says that the simulator can be chosen such that no matter which protocol we immerse the algorithm the final outputs of the protocol with respectively the algorithm and the distinguisher are virtually indistinguishable.

If $x \in L_R$ let $W(x)$ be *the* witness such that $R(x, W(x))$, else let $W(x) = \perp$. We have

Lemma 1

$$\begin{aligned}
&\forall \alpha > 0 \forall A \in PPT \exists S \in PPT \forall B, D \in PPT \exists K \forall k > K \forall z :\\
&|P((s,t) \leftarrow B^{\mathcal{O}}(z); ((x_1, p_1), \dots, (x_l, p_l), u) \leftarrow A^{\mathcal{O}}(s) :\\
&\qquad x_1, \dots, x_l \in L \wedge \overline{\mathcal{O}_B} \cap \overline{\mathcal{O}_A} = \emptyset\\
&\qquad \Rightarrow D^{\mathcal{O}}(t, u, (x_1, p_1, W(x_1)), \dots, (x_l, p_l, W(x_l))) = 1)\\
&-P((s,t) \leftarrow B^{\mathcal{O}}(z); ((x_1, p_1, w_1), \dots, (x_l, p_l, w_l), u) \leftarrow S^{\mathcal{O}}(s) :\\
&\qquad x_1, \dots, x_l \in L \wedge \overline{\mathcal{O}_B} \cap \overline{\mathcal{O}_S} = \emptyset\\
&\qquad \Rightarrow D^{\mathcal{O}}(t, u, (x_1, p_1, w_1), \dots, (x_l, p_l, w_l)) = 1)| < \frac{1}{k^\alpha},
\end{aligned}$$

where $\overline{\mathcal{O}_B}$ denotes the set of queries made by B, while $\overline{\mathcal{O}_A}$ and $\overline{\mathcal{O}_S}$ denote the sets of queries made by the respective algorithms to the random oracle as well as the queries used in the proofs. And where we demand that the proofs $p_1, \dots, p_l$ produced by A are valid.

Furthermore, the algorithm S can be uniquely determined from A, t_A, α and an extractor E. We write $S^{A,\alpha}$ when we want to emphasize this.

We note that there are some limitations in the lemma above. First of all, albeit the probability of differentiating between the two scenarios can be made arbitrarily small, it is not negligible. Second we note that the algorithm may not ask oracle queries already made previously in the protocol, neither use these queries directly in the proofs. The reason for these restrictions stem from the fact that the simulator we present simply runs A several times giving it different oracle answers, trying to make it make two different proofs for the same element. This will fail if A can somehow use a premade proof made by B, and it will fail if A can somehow from the information from B detect that it is not getting correct oracle answers.

Proof. The idea is the following: The simulator S runs a copy of A answering oracle queries using the random oracle. A produces elements $x_1, \dots, x_l$, proofs $p_1, \dots, p_l$ and some extra output u. This is the output that S will use, however, the simulator needs to find corresponding witnesses to the valid proofs.

Call this execution of A for the main copy of A. In addition, whenever the main copy makes an oracle query q, S runs many extra copies of A from the state A was in at the time it made the query. In the extra copies oracle queries are answered at random by S instead of using the random oracle. This way the extra copies of A get answers on q that differ from the real random oracles answer on q.

Looking at a situation where A has just made an oracle query q there are two possibilities: There is a low probability that the answer is used by A in a proof in the end, and in that case we do not need to worry about it because it probably does not appear in the main execution. Alternatively there could be a high probability that it is used by A in a valid proof, in which case we can hope that one of the many extra copies also results in it being used in a valid proof. In this case, since the query answers are different in the extra copies, we end up with two proofs using two different challenges but with the same initial message. By the special soundness property of the original 3-move interactive proof system we can extract a witness from the proofs if the query is used in the main execution too. This way we have witnesses $w_1, \ldots, w_l$ to pass along with the main executions output, and of course the distribution of x_i's and p_i's is completely identical to that of a real execution of A.

Let us specify in details how S acts on input s. Along the way S maintains some sets or lists keeping track of information pertaining to the oracle queries it makes.

<u>S on input s.</u>

```
Set $\overline{\mathcal{O}} = \emptyset$.
Run $A$ on $s$ handling oracle queries and halting as below.

If $A$ makes an oracle query $q$ do:
  Save the entire state of $A$.
  Set $\overline{\mathcal{O}_q} = \emptyset$.
  For $j = 1$ to $k^{\lceil t_A + \alpha + 1 \rceil}$ do:
    Select at random a string $e'$ of suitable length.
    Run $A$ from the saved state, answering the query $q$ with $e'$,
    and answering subsequent oracle queries at random.
    Check in the resulting output $((x'_1, p'_1), \ldots, (x'_{l'}, p'_{l'}), u')$
    whether $q$ and $e'$ have been used in a pair $(x', p')$.
    In that case if the proof is valid let $\overline{\mathcal{O}_q} = \{(e', x', p')\}$.
  Query the random oracle for an answer $e$ to $q$.
  Let $\overline{\mathcal{O}} = \overline{\mathcal{O}} \cup (q, e)$.
  Continue the run of $A$ from the saved state and with answer $e$ to
  its query.

If $A$ terminates outputting $((x_1, p_1), \ldots, (x_l, p_l), u)$ do:
  For $j = 1$ to $l$ do:
    Check whether $\overline{O_{q_j}} \neq \emptyset$.
    In that case set $w_j = E(x_j, a, e, e', z, z')$ where $a$ is the initial
    message in the proof, $e, e'$ are the respective oracle answers in
    the main copy of $A$ and in $\overline{O_{q_j}}$, and $z, z'$ are the answers
    completing the proofs.
    If any of these two checks failed set $w_j = \perp$.
  Output $((x_1, p_1, w_1), \ldots, (x_l, p_l, w_l), u)$ and halt.
```

The argument for this S being the kind of simulator we seeking goes like this: Look at a situation where A has just made an oracle query. We are interested in the probability of A putting the oracle answer to good use in the end, i.e. its output includes an element $x \in L$ and a valid proof p for $x \in L_R$. We have two situations, one where this probability is less than or equal to $\frac{1}{2k^{t_A+\alpha}}$ and one where it is above $\frac{1}{2k^{t_A+\alpha}}$.

Since A can make no more than k^{t_A} queries the probability of any of the oracle queries with less than or equal to $\frac{1}{2k^{t_A+\alpha}}$ probability of being used, is actually used is less than $\frac{1}{2k^{\alpha}}$.

We can therefore assume that only oracle queries with usage probability larger than $\frac{1}{2k^{t_A+\alpha}}$ is used in the proofs for elements $x \in L$ in the final output of the main copy of A. However, since we ran $k^{\lceil t_A+\alpha+1 \rceil}$ copies of A from this situation, we thus have more than $1 - e^{-k}$ chance of the same oracle query having been used in a valid proof in one of the extra copies of A. In this case, according to the special soundness we can extract a witness with overwhelming probability.

It is obvious that the distribution of elements $x_1, \ldots, x_l$ and proofs $p_1, \ldots, p_l$ from algorithms A and S respectively is completely identical. Moreover, from the argument above we have that with probability higher than $1 - \frac{1}{k^{\alpha}}$ S also passes along witnesses $w_1, \ldots, w_l$ that match $W(x_1), \ldots, W(x_l)$ for all of the $x_1, \ldots, x_l$ belonging to L. This covers the main part of the lemma.

To conclude the proof we note that S described here is based on A, t_A, α and an extractor E. If we fix E for the interactive proof system we thus get a standard simulator $S^{A,\alpha}$ that fits the lemma. $\square$

We remark that a slight modification of the above proof also demonstrates the truth of Lemma 1 where the 3-move interactive proof system does not have the special soundness property but instead a more relaxed condition is satisfied: getting at least a polynomial number of different proofs based on he same initial message one can compute a witness.

Another possible relaxation of the special soundness criterion is one where we are only interested in some partial information about the witness w. In that case it is sufficient that the extractor in the special soundness definition extract this partial information from the witness, i.e. some function value $f(w)$ is computed rather than the entire witness.

4 Damgård-Jurik voting

We present the first example of a 3-move interactive proof system. It will be used in a threshold version of a generalization of Paillier's cryptosystem.

4.1 Equality of discrete logarithms proof

Let L be the language of sextuples (n, s, u, u', v, v') such that n is a product of two large safe primes $p = 2p' + 1, q' = 2q' + 1$, such that s is a small positive integer, and such that u, u', v, v' are squares in $Z^*_{n^{s+1}}$ with v being a generator for the group of squares. Let furthermore R be the relation that on input $((n, s, v, v', u, u'), y)$ is true if and only if $n, s \in Z_+ \wedge v, v', u, u' \in Z^*_{n^{s+1}} \wedge v' \equiv v^y \bmod n^{s+1} \wedge u' \equiv u^y \bmod n^{s+1}$. Finally let t be some secondary security parameter such that $2^t < p', q'$ (If we wish to continue working with only one security parameter we can let the size of t depend on k, say $t = \lceil k/3 \rceil$). We present a 3-move honest verifier zero-knowledge interactive proof system with the special soundness property for L_R relative to L.

3-move interactive proof system for L_R relative to L
Input: n, s, u, u', v, v'.
Private input for the prover: y such that $R((n, s, u, u', v, v'), y)$.

1. The prover selects an $(s + 1)k + 2t$-bit number r at random. It sends $a = u^r \bmod n^{s+1}$ and $b = v^r \bmod n^{s+1}$ to the verifier.

2. The verifier sends a random a t-bit challenge e to the prover.

3. The prover answers the challenge by sending $z = r + ey$ to the verifier.

4. The verifier accepts if and only if $u^z \equiv av'^e \bmod n^{s+1}$ and $v^z \equiv bv'^e \bmod n^{s+1}$.

Using the Fiat-Shamir heuristic this can be made a non-interactive zero-knowledge proof system for L_R in L in the random oracle model. Note that in the non-interactive proof the verifier acts deterministically, and therefore anybody verifying the proof will agree on whether it is acceptable or not.

4.2 Threshold generalized Paillier encryption

We present the (ω, N)-threshold generalized Paillier encryption scheme from [DJ01]. It uses a (small) positive integer s as a parameter to describe how long ciphertexts we allow.

Key Generation

Input: s.

Generate a k-bit integer $n = pq$, where p and q are large safe primes, i.e. $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are also primes. Choose $d \in \{1, \dots, n^s p'q'\}$ such that $d \equiv 0 \bmod p'q'$ and $d \equiv 1 \bmod n^s$.

This d is the secret key of the encryption scheme. Since we want a threshold encryption scheme we secret share it amongst the authorities. Select a polynomial $f(X) = \sum_{i=0}^{\omega-1} a_i X^i \bmod n^s p'q'$ at random by setting $a_0 = d$ and picking $a_1, \dots, a_{\omega-1}$ at random from $\{0, \dots, n^s p'q' - 1\}$. Note that $f(0) = d$. The shares are $s_1 = f(1), \dots, s_N = f(N)$.

We want to enable each authority to prove that he has used the correct share when decrypting. This is done using the equality of discrete logarithms proof presented before. To set up the possibility of such a proof we choose at random a square v in $Z^*_{n^{s+1}}$ and let $v_1 = v^{\Delta s_1} \bmod n^{s+1}, \dots, v_N = v^{\Delta s_N} \bmod n^{s+1}$, where $\Delta = N!$.

The public key is now $pk = (n, s, v, v_1, \dots, v_N)$ while the private keys are $s_1, \dots, s_N$.

Encryption

Input: Public key pk and plaintext $m \in Z_{n^s}$.

Select at random $r \in Z^*_n$ and let the ciphertext be given by $c = (1 + n)^m r^{n^s} \bmod n^{s+1}$.

Decryption

Input: Public key pk, ciphertext $c = (1 + n)^m r^{n^s} \bmod n^{s+1}$, and secret keys $s_1, \dots, s_N$ known to the respective authorities.

Each authority computes $c_i = c^{2\Delta s_i}$ and proves non-interactively that $\log_v(v_i) \equiv \log_{c^4}(c_i^2) \bmod \phi(n^{s+1})$ using the non-interactive version of the proof of equality of discrete logarithms protocol mentioned before. Both the share of the decryption and the proof of it having been correctly formed are published on the message board.

Using Lagrange interpolation ω correct shares enables us to decrypt the ciphertext. Let T be the set of the first ω shares with valid proofs and compute

$$c' = \prod_{i \in T} c_i^{2\lambda_{0,i}} \equiv c^{\sum_{i \in T} 4\Delta s_i \lambda^T_{0,i}} \equiv c^{4d\Delta^2} \equiv (1+n)^{m4d\Delta^2} \bmod n^{s+1}$$

where $\lambda^T_{0,i} = \Delta \prod_{i' \in T \setminus i} \frac{-i}{i-i'}$.

From this we can retrieve m using the technique from [DJ01].

An important property of this encryption scheme is that it is homomorphic. Given two ciphertexts c_1 and c_2 encrypting m_1 and m_2 one can obtain a ciphertext $c_3 = c_1 c_2 \bmod n^{s+1}$ encrypting $m_1 + m_2 \bmod n^s$.

The encryption is shown in [DJ01] to be semantically secure provided the Decisional Composite Residuosity Assumption described below holds for products of safe primes.

Decisional Composite Residuosity Assumption
Choose n at random among the k-bit integers that are products of two large primes. The distribution of $r^n \bmod n^2$ where r is chosen at random from Z_n^* is computationally indistinguishable from the uniform distribution on $Z_{n^2}^*$.

4.3 Encryption of 0 or 1 proof

We will later look at votes being 0 or 1 and will therefore need a non-interactive proof system for a ciphertext c encrypting 0 or a 1. More precisely let L be the language of pairs (pk, c) where pk is a public key chosen as above, and where $c \in Z_{n^{s+1}}^*$. Moreover, let R be a relation computable in polynomial time such that $R((pk, c), r)$ if and only if $r \in Z_n^*$ and $c = r^{n^s} \bmod n^{s+1} \vee c = (1+n)r^{n^s} \bmod n^{s+1}$. We seek a non-interactive zero-knowledge proof for L_R in L. This can be obtained in the random oracle model using the Fiat-Shamir heuristic on the proof system below.

3-move interactive proof system for L_R relative to L
Input: Public key pk chosen as above, ciphertext $c \in Z_{n^{s+1}}^*$, and a secondary security parameter t such that $2^t < p', q'$.
Private input for the prover: $r \in Z_N^*$ such that $R((pk, c), r)$.

1. Let $c_0 \equiv c \bmod n^{s+1}$ and $c_1 \equiv (1+n)^{-1} c \bmod n^{s+1}$. Let b be the bit such that c_b is an encryption of 0. The prover starts by choosing $z_{\bar{b}} \leftarrow Z_n^*$, $e_{\bar{b}} \leftarrow \{0, \ldots, 2^t - 1\}$ and setting $a_{\bar{b}} = z_{\bar{b}}^{n^s} c_{\bar{b}}^{-e_{\bar{b}}} \bmod n^{s+1}$. Together $(a_{\bar{b}}, e_{\bar{b}}, z_{\bar{b}})$ constitute a simulated proof for $c_{\bar{b}}$ being an encryption of 0. Next the prover selects $r' \leftarrow Z_n^*$ and sets $a_b = r'^{n^s} \bmod n^{s+1}$. The prover now sends (a_0, a_1) to the verifier.

2. The verifier selects at random a non-negative integer ϵ less than 2^t and returns this challenge to the prover.

3. The prover sets $e_b = \epsilon - e_{\bar{b}} \bmod 2^t$ and lets $z_b \equiv r' r_b^e \bmod n$. (a_b, e_b, z_b) constitute a real proof that c_b is an encryption of 0. The prover answers the challenge by sending (e_0, e_1, z_0, z_1) to the verifier.

4. The verifier accepts if and only if $z_0^{n^s} \equiv a_0 c_0^{e_0} \bmod n^{s+1}$, $z_1^{n^s} \equiv a_1 c_1^{e_1} \bmod n^{s+1}$ and $\epsilon = e_0 + e_1 \bmod 2^t$.

This proof system has the special soundness property and is honest verifier zero-knowledge. Note that in the non-interactive version, the verifier acts deterministically and therefore anybody will agree on whether to accept or reject the proof.

4.4 The voting scheme

Threshold generalized Paillier encryption can be used in a voting scheme. For simplicity, we describe here a version where voters can only vote yes or no. A voter votes yes or no by posting respectively an encryption of 1 or 0 on the message board. Taking advantage of the homomorphic property of cryptosystem, the authorities obtain an encryption of the entire result by multiplying all the individual votes together. Now they jointly decrypt this to get the final tally. To avoid a voter encrypting illegal votes we demand that along with the vote he also publishes a proof as described above that the vote is an encryption of 0 or 1.

The voting scheme

Parties There are three types of parties: M voters $V_1, \dots, V_M$, N authorities $A_1, \dots, A_N$, and Q independent verifiers $P_1 \dots, P_Q$. The voters start with inputs $x_1, \dots, x_M$, which are the votes that they intend to cast.

Key Setup We assume that security parameters k and t as well as a suitable size parameter s have been selected in advance. Now generate keys for threshold generalized Paillier encryption: The public key $(n, s, v, v_1, \dots, v_N)$ is published at the message board for all to see. The shares for the private key, $s_1, \dots, s_N$ are given secretly to the corresponding authorities $A_1, \dots, A_N$.

Voting Each voter encrypts his vote and attaches a proof that it encrypts 0 or 1. The resulting vote consisting of ciphertext and proof is published on the message board.

Tallying Each authority reads the posts on the message board submitted by the voters, and checks for each voter that he has only posted one ciphertext belonging to $Z^*_{n^{s+1}}$ and that it is accompanied by a valid proof of being an encryption of either 0 or 1. It notes the number of valid votes, and it calculates the product of the ciphertexts in the valid votes. The number of voters as well as the resulting product of ciphertexts, c, is published on the message board.

Now each authority computes its share of the decryption of the result by letting $c_i = c^{2\Delta s_i} \bmod n^{s+1}$. This share is also posted on the message board together with a proof that $\log_{c^2}(c_i^4) \equiv \log_v(v_i) \bmod \phi(n)$.

Having completed these two steps all authorities now find the first ω authorities who have posted a decryption share c_i together with a valid proof of it being legal. Using these shares each authority now decrypts the product of the ciphertexts, c, to get the result of the election.

The authorities post the result of the election on the message board. Finally, from the message board all parties read off the result as the majority decision of the authorities and output this. The result consists of the number of valid votes and the number of yes votes.

The voting scheme can be extended to accommodate elections with more than two choices. For the purpose of this paper, however, it suffices to over only the simple case with two choices. The more general case is treated quite similarly.

5 Security of the Voting Scheme

We compare the voting scheme with an ideal process. In the ideal process, each voter hands his vote to a trusted party over a secure channel. After that the trusted party tallies the votes and sends the result to each party. We demonstrate that for any nonadaptive adversary attacking the voting scheme there exists a simulator in the ideal process model such that the results and outputs of the adversary, respectively simulator, are indistinguishable. This shows that the voting protocol is secure according to the multi-party computation security definition in [Can00].

Before formulating the above in a precise theorem, we clarify the model in which we are working. We imagine there is an adversary able to corrupt up to $\omega - 1$ authorities and as many voters and independent verifiers as he wants to. Without reducing the adversary's strength we may assume that these are corrupted from the start of the protocol and that the adversary simply gets a list of corrupted parties as input from the start. The corrupted parties are completely under the adversary's control, he gains all information they get during the execution of the protocol, and they output messages at his whim. In addition, the adversary may get some auxiliary input from the start, intuitively representing information he may have gained by corrupting parties and information he may have gathered from secondary resources. The only disadvantage he has is that the corrupted parties are fixed and cannot be changed during the protocol execution.

We have the following model of the network in which the election takes place:

Insecure channels The adversary sees all communication between the parties.

Identity Each party has a unique ID number known to all parties.

Authentication Each message sent on the network come together with a tag, which identifies the sender. It is a convenience to assume that the network is authenticated but there is no loss of generality in doing so. We refer the reader to for instance [BCK98] for information on how to authenticate networks in a modular way fitting with the framework we present here.

Non-blocking All messages sent arrive at the intended receiver, the adversary cannot stop messages from arriving.

Synchronous Execution of protocols is divided into rounds. During a round each party is activated once, and during this activation it will be able to send messages to other parties. After its activation the party is not activated again until next round. Control passes onto another party.

Rushing The adversary determines when to activate parties within each round. In particular it can let all corrupted parties wait until the end of the round thereby potentially getting an advantage by knowing the messages sent by uncorrupted parties before taking any action itself.

To describe security of a protocol we compare two scenarios:

In the first scenario called the real life model the protocol is conducted as normal with the adversary possibly corrupting some parties. The goal of the adversary is to alter the output of honest parties or to gather some knowledge, which was supposed to be inaccessible.

The second scenario is called the ideal process. In this process, we can think of the parties as handing their inputs to the protocol to a trusted party over secure channels. The trusted party figures out what should be the result of the protocol and answers each party. Each party simply outputs this answer as his final output in the protocol. The adversary can still corrupt parties before the evaluation altering some parties' input to the trusted party. Moreover, it can still alter corrupted parties' output after the secure evaluation. However, the core of the protocol computation made by the trusted party remains impossible to attack.

We say that a protocol is secure if for any adversary conducting an attack in the real life model there is an adversary attacking the ideal process such that the combined outputs of the honest parties and the adversary in the respective two scenarios are indistinguishable.

One advantage of this definition of security is that it is allows modular composition. If some sub-protocol can be proven secure, one can replace this part of the protocol with an ideal process. The resulting main protocol's security properties remain unchanged under this substitution.

We shall see that the voting scheme presented in this paper is secure. In other words, we compare the following two experiments:

Real life experiment

Each voter $V_1, \ldots, V_M$ start out with a choice $x_1, \ldots, x_M$ of the vote he wants to cast. The adversary starts out with a set of corrupted voters, less than ω corrupted authorities, and some corrupted independent verifiers plus some string z.

1. Election keys are generated. The public key $(n, s, v, v_1, \ldots, v_N)$ is placed on the message board. Shares $s_1, \ldots, s_N$ of the private key d are given to the respective authorities.

2. Each honest voter computes his vote and posts it on the message board. The adversary selects the messages of the corrupted voters to be posted on the message board.

3. The honest authorities calculate the product c of the valid votes, their shares for decrypting c along with proofs that the shares are correct, and publish all this on the message board.
   The adversary algorithm may decide on some messages to be posted by the corrupted authorities.

4. Each honest authority computes from the first ω valid shares the result of the election and posts it on the message board.

5. Each honest party outputs the result.
   The adversary makes some output of his choice.

In the experiment described above the adversary is only allowed to take some specific actions at certain times. However, it would not make the adversary stronger to allow it further freedom in its choices so there is no loss of generality in this restriction.

Ideal process experiment

Again each voter starts out with a choice, and the adversary S starts out with a set of corrupted parties and a string z. We shall from now on call S for the simulator since it will simulate a real life model adversary A. We describe S along the way.

1. S runs the key generation protocol for the threshold version of generalized Paillier encryption. During the key generation S learns the secret key d and is therefore able to decrypt messages in the following part of the experiment.

2. S does not know the choices of the honest voters, but makes a random choice, 0 or 1. These choices are encrypted to produce real votes, which are given to A as if they were posted on the message board.
   A is invoked as had it seen the public key $(n, s, v, v_1, \ldots, v_N)$ on the message board, some secret key shares s_i with corrupted authorities, the votes made by honest voters on the message board, and as had it started out with the string z. A decides whether to post some messages for corrupted voters on the message board. Those of the messages corresponding to valid votes are decrypted by S. S changes the input of the corrupted voters to be those choices, while corrupted voters for which A submits invalid votes are instructed not to submit votes to the trusted party.

3. The trusted party now receives the votes and sends the tally to all parties. S learns the result from one of the corrupted parties[2] .

4. Let m be the real result and let m' be the result from the random choices selected by S and the votes made by A. In order to continue the simulation S must trick A to believe that the product of valid votes decrypt to m where in reality they decrypt to m'. For this purpose we choose d' in $\{0, \ldots, n^s p'q' - 1\}$ such that $d' \equiv 0 \bmod p'q'$ and $d' \equiv (\frac{m}{m'}) \bmod n^s$. Now secret key shares $s'_1, \ldots, s'_N$ for d' are selected such that they fit with the shares, s_i's, that A already knows, and such that the polynomial intersects d' in 0. With these shares S produces for honest authorities shares $c_i = c^{2\Delta s'_i} \bmod n^{s+1}$ and simulates proofs for $\log_{c^4}(c_i^2) \equiv \log_v(v_i) \bmod \phi(n^{s+1})$. This simulation can be done because S controls the random oracle seen by the simulated A. S can just select values at random when making up the random oracle's responses.
   Once again A is enacted and produces messages on behalf of the corrupted authorities.

[2]We assume there is at least one corrupted party. Otherwise, security is obvious.

5. We see that the ω first valid shares c_i give

$$\begin{aligned} c' = & \prod_{i \in S \setminus i} c_i^{2\lambda_{0,i}^S} \bmod n^{s+1} \\ \equiv & c^{\sum_{i \in S \setminus i} 4\Delta s_i' \lambda_{0,i}^S} \bmod n^{s+1} \\ \equiv & c^{4\Delta^2 d'} \bmod n^{s+1} \equiv (1+n)^{4\Delta^2 i} \bmod n^{s+1}, \end{aligned}$$

which means that the "shares" of c' combine to the result i.
A is given the result and produces its final output.

We wish to demonstrate that the distribution of the pairs of the result and the output of the adversary in the real life model is indistinguishable from the result and output of the simulator in the ideal process model. In other words, we want to demonstrate:

Theorem 1

$$\begin{aligned} &\forall A \in ITM \exists S \in ITM \forall D \in PPT \forall \delta > 0 \exists K \forall k > K \forall x_1, \ldots, x_M \in \{0,1\} \forall z : \\ &\quad P((\text{result}, \text{output}) \leftarrow Exp_{RLM,A^{\mathcal{O}}}(x_1, \ldots, x_M, z) : \\ &\qquad D^{\mathcal{O}}(\text{result}, \text{output}) = 1) \\ &\quad - P((\text{result}, \text{output}) \leftarrow Exp_{IP,S^{\mathcal{O}}}(x_1, \ldots, x_M, z) : \\ &\qquad D^{\mathcal{O}'}(\text{result}, \text{output}) = 1) < \frac{1}{k^\delta}. \end{aligned}$$

where $\mathcal{O}'$ is $\mathcal{O}$ modified such that it fits with the oracle answers made by S when simulating proofs.

Proof. Let us first look at the real life model experiment and modify it into another experiment $Exp_{RLM',A^{\mathcal{O}}}$, which is indistinguishable from $Exp_{RLM,A^{\mathcal{O}}}$ [3].

The correctness of the decryption shares made by the honest authorities is demonstrated by equality of logarithms proofs, which are non-interactive zero-knowledge proofs in the random oracle model. Modifying the random oracle correspondingly we can therefore simulate the proofs of the honest authorities producing the correct decryption shares.

The elements $v, v_1, \ldots, v_N$ used to demonstrate the correctness of the decryption shares can be created without knowledge of the decryption key d in the following way: Select for $\omega - 1$ authorities, including the corrupted authorities, at random elements $s_i \in \{0, \ldots, \lfloor \frac{n^{s+1}}{4} \rfloor\}$. This is indistinguishable from choosing them at random in $\{0, \ldots, n^s p' q'\}$. Select a square v such that it has a known plaintext, i.e. $v = (1+n)^m r^{2n^s} \bmod n^{s+1}$. Let $s_0 = d$. Now,

[3]To reduce notation a little we do not write the inputs $(x_1, \ldots, x_M, z)$ to the experiments

even though we do not know d, we do know that $v^d \equiv (1+n)^m \bmod n^{s+1}$. Let T be the set of the ω indices $\{0, i_1, \ldots, i_{\omega-1}\}$, and let

$$\lambda_{j,i}^T = \Delta \prod_{i' \in T \setminus i} \frac{j-i}{i-i'}$$

then the remaining v_j's can be computed as

$$v_j = \prod_{i \in T} (v^{s_i})^{\lambda_{j,i}^T} \bmod n^{s+1}.$$

This gives us $v, v_1, \ldots, v_N$ distributed the right way without having used d to compute them.

The proofs for validity of the votes for the honest voters can also be simulated without changing the result-output distribution of the experiment distinguishably.

Finally the decryption shares for the honest authorities can be constructed from the result m of the election instead without using their shares of the secret key. This is done in a similar way as $v, v_1, \ldots, v_N$ were made above, using the same selection of s_i's and T. Even though we do not know d we do know that $c^{2s_0} = (1+n)^{2m} \bmod n^{s+1}$. For j's not in T we can therefore compute shares as

$$c_j = \prod_{i \in T} (c^{2s_i})^{\lambda_{j,i}^T} \bmod n^{s+1}.$$

We let the experiment $Exp_{RLM',A^{\mathcal{O}}}$ be $Exp_{RLM,A^{\mathcal{O}}}$ modified in this manner.

We shift our attention to the ideal process. Here we modify S into S' such that $Exp_{IP,S^{\mathcal{O}}}$ is indistinguishable from $Exp_{IP,S'^{\mathcal{O}}}$. The decryption shares correctness is already simulated. The elements $v, v_1, \ldots, v_N$ can be simulated without knowing d as mentioned before when we modified the real life model. Furthermore, also in this model we can simulate the proofs of correctness of the votes for the honest voters. Finally, we can compute the shares from the result as described before, this time just pretending that $c^{2s_0} = (1+n)^{2m} \bmod n^{s+1}$.

It is immediately clear that the results in $Exp_{RLM',A^{\mathcal{O}}}$ and $Exp_{IP,S'^{\mathcal{O}}}$ are indistinguishably distributed. We still need to argue that the outputs of the adversaries look the same though.

The modifications to the two models make them very similar. Keys are generated the same way, votes are generated the same way, the decryption process works the same way and the decryption shares are calculated the same way. The difference between $Exp_{RLM',A^{\mathcal{O}}}$ and $Exp_{IP,S'^{\mathcal{O}}}$ is that in the ideal process the encrypted votes seen by the adversary do not add up to the result. We want to use the semantic security of the underlying cryptosystem to demonstrate that the outputs of the adversary A in the two models are indistinguishable. However, this cannot be done directly since the result contains information concerning what is encrypted in the votes made by the adversary, information that could normally only be gathered using the decryption key. Therefore we would need the cryptosystem to be non-malleable to make such an argument, and the notion of non-malleability is stronger than the notion of semantic security.

Instead, imagine there was some non-negligible difference between the two modified experiments. Say

$$\begin{aligned}
&\exists D \in PPT \exists \beta > 0 \forall K \exists k > K \exists x_1, \ldots, x_M \in \{0,1\}, z : \\
&P((\text{result}, \text{output}) \leftarrow Exp_{RLM',A^{\mathcal{O}}} : D^{\mathcal{O}'}(\text{result}, \text{output}) = 1) \\
&-P((\text{result}, \text{output}) \leftarrow Exp_{IP,S'} : D^{\mathcal{O}'}(\text{result}, \text{output}) = 1) > \frac{3}{k^\beta}
\end{aligned}$$

where $\mathcal{O}'$ is $\mathcal{O}'$ modified such that it fits with the simulated proofs produced in the two experiments.

We note that the part of A producing votes and proofs for validity of those votes cannot use queries made previously in the protocol. They are all used in proofs and thus posted on the message board, so there is no reason to ask the oracle about them again. And if it copied proofs directly it runs into the problem that the queries contain the identity of the voters, and therefore they cannot be used for the corrupted voters. Therefore, according to Lemma 1 we can choose a simulator $S^{A,\beta}$ for the part of A producing votes such that in addition to the proofs of validity of the votes there are accompanying witnesses for the votes being valid. Those witnesses, the randomness involved in the encryptions, can be used to reveal the underlying yes or no choices in the valid votes produced by the adversary. If we call the modified real life experiment where the simulator is to produce the votes instead of A for $Exp_{RLM',A^{\mathcal{O}}[S^{A,\beta}]}$ we get that no distinguisher has a higher chance of differentiating between $Exp_{RLM',A^{\mathcal{O}}[S^{A,\beta}]}$ and $Exp_{RLM',A^{\mathcal{O}}}$ than $\frac{1}{k^\beta}$.

The exact same simulator $S^{A,\beta}$ can also be used in connection with the ideal process model. Here we modify S' such that it runs $S^{A,\beta}$ instead of A when making votes and proofs of validity of the votes. Now, instead of decrypting the valid votes to figure out what the content is, it can use the witnesses to figure that out. Calling this experiment for $Exp_{IP,S'^{\mathcal{O}}[S^{A,\beta}]}$ we have from Lemma 1 that the advantage of any distinguisher trying to distinguish $Exp_{IP,S'^{\mathcal{O}}[S^{A,\beta}]}$ and $Exp_{IP,S'^{\mathcal{O}}}$ is less than $\frac{1}{k^\beta}$.

Because we are using the witnesses to extract the contents of the votes made by A we do not at all need to use the decryption key d. The result can be computed using only the witnesses. We now have that it is possible to distinguish the two experiments $Exp_{IP,S'^{\mathcal{O}}[S^{A,\beta}]}$ and $Exp_{RLM',A^{\mathcal{O}}[S^{A,\beta}]}$ with an advantage of at least $\frac{1}{k^\beta}$. However, this contradicts the semantic security of the underlying cryptosystem saying that the advantage should be negligible. □

We made this proof just in the case where votes are represented as 0 or 1. Damgård and Jurik's article generalize this to the case where one can vote for t out of L choices. We note that the proof generalizes also to this scenario with virtually the same arguments.

6 Acknowledgments

We would like to thank Ivan Damgård for suggesting the possibility of the voting scheme being secure according to Canetti's definition and for useful discussions.

References

[BCK98] M. Bellare, R. Canetti, and H. Krawczyk. Modular approach to the design and analysis of key exchange protocols. In *proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98) (New York, May 23–26 1998)*, pages 419–428, 1998.

[Can00] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CDN01] R. Cramer, I. Damgard, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *proceedings of EUROCRYPT '01, LNCS series, volume 2045*, pages 280–299, 2001.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally eficient multi-authority election scheme. In *proceedings of EUROCRYPT '97, LNCS series, volume 1233*, pages 103–118, 1997.

[DJ01] I. Damgård and M.J. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001, LNCS series, volume 1992*, 2001.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game, or A completeness theorem for protocols with honest majority. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1987.

[PS00] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[Sho00] V. Shoup. Practical threshold signatures. In *Advances in Cryptology — EUROCRYPT '00, volume 1807 of Lecture Notes in Computer Science, pages 207-220.*, 2000.