Using non-interactive proofs to achieve independence efficiently and securely

Rosario Gennaro
Laboratory for Computer Science
Massachusetts Institute of Technology
rosario@theory.lcs.mit.edu

November 4, 1994

Abstract

Independence or *simultaneous broadcast* is a fundamental tool to achieve security in fault tolerant distributed computing. It allows n players to commit to independently chosen values. In this paper we present a constant round protocol to perform this task. Previous solutions were all $O(\log n)$ rounds. In the process we develop a new and stronger formal definition from this problem.

As an example of the importance of independence in distributed protocols, we show an attack on the Sako-Kilian election scheme presented at CRYPTO 94 made possible by the protocol failure on achieving independence. Using our techniques we will show how to modify the scheme to make it secure.

1 Introduction

Independence is a fundamental tool to achieve security in fault tolerant distributed protocols. In this paper we present improved results based on a careful exploitation of the properties of non-interactive proofs [2]. In particular we will exhibit the first constant round protocol for the problem of *simultaneous broadcast* (previous solutions were $O(\log n)$ rounds where n is the number of processors in the system). In the process we will develop a new and stronger formal definition for this problem. Finally as a practical example of the importance of achieving independence in distributed protocol we show an attack to the Sako-Kilian election scheme [10] made possible by the protocol failure on achieving independence. Using our techniques we will show how to fix this problem and make Sako-Kilian scheme secure.

1.1 Why independence?

The problem of independence in distributed protocols has been put forward by Chor and Rabin in [3] and subsequently by Dolev, Dwork and Naor in [5].

Informally this means that we are looking for protocols in which two or more parties announce some values and we want to prevent any of them to correlate in *any manner* his/her own values to the values that the other parties have announced. To explain the concept of independence let us present some motivating scenarios first.

Example 1 *Contract bidding* [1]: The city of Cambridge decides to sell an estate along the Charles River and invites potential inquirers to bid for the deal. The city publishes a public-key encryption scheme E for interested people to send their bid. Both Harvard University and MIT are interested in buying the estate. MIT places its bid by sending $m_M = E(b_M)$ over an insecure phone line. Harvard is tapping the line and overhears the MIT bid. In spite of the security of the encryption scheme E, now Harvard is able to at least tie the MIT bid, by sending over $m_H = m_M$. Even requiring Harvard to send a different cyphertext would not solve the problem since it could still be possible to produce a cyphertext $m_H \neq m_M$ that decrypts to a lower bid.

Example 2 *Coin flipping* : n players decide to flip together a common and random coin b. A way to do this would be for each player P_i to broadcast a bit b_i and then set b to be the XOR of all the broadcasted bits. This is clearly a bad idea since the last player that broadcasts his/her bit decides the value of the coin. Similarly having all the players commit to their bits by posting $E(b_i)$, where E is a encryption scheme and then having them decommit, does not solve the problem either since, as we saw in the previous example, encryption alone does not guarantee independence. For example it could be possible for the last player after seeing $E(b_1), \ldots, E(b_{n-1})$ to broadcast m_n such that at decommiting time $b = 0$. Worse a player could decide to not decommit his/her bit if he/she realizes that the outcome of the coin toss is not the desired one.

Example 3 *Electronic elections*: In the future we may think of running elections on the network. A possible way of doing this would be to have a center collecting the votes sent to it in some encrypted form. However we need to make sure that casted votes are independent, i.e., that seeing the cyphertext of one voter does not influence the actions of another voter.

The examples above show that encryption is necessary to this task, but not sufficient. The reason is that the definition of *semantic security* for cryptosystems [9] guarantees that given a cyphertext we cannot learn anything about the corresponding cleartext. However in distributed settings this is not enough: we want the cyphertext to be completely useless. As pointed out in [5] independence is clearly an extension of the concept of semantic security.

[1] This example has been adapted from a similar one described in [5]

1.2 Previous work

Chor and Rabin in [3] put forward a first formal definition for the problem of *simultaneous broadcast*: a protocol that allows n players to independently announce n bits. Their solution requires $O(\log n)$ rounds to complete. Clearly this protocol solves the coin flipping problem.

Subsequently in a very nice paper Dolev, Dwork and Naor in a quite different setting addressed the problem of independence in encrypted communication and zero-knowledge proofs. They call their property *non-malleability*. They present non-malleable bit commitment schemes, encryption schemes and zero-knowledge interactions. In particular they solve the contract bidding problem.

1.3 Our contribution

In this paper, drawing from ideas of our predecessors, we improve on those previous results. More in detail:

1. We describe a *constant* round protocol for simultaneous broadcast. In doing that we will heavily rely on the properties of non-interactive zero-knowledge proofs as introduced by Blum, De Santis, Micali and Persiano in [2].

2. We refine the definition of independence to a stronger one. In fact both in [3] and in [5] they define the independence property with respect to a polynomial-time bounded observer. That is no polynomial-time Turing machine that has access to a random sample of successful executions of the protocol is able to detect any correlation among the committed values of the players. In particular neither the players or the adversary are able to do so. Our definition instead, as we will see shortly, propose something that we would call *statistical independence*, i.e., we require that an observer of *any* computational power who is given a polynomial size random sample of successful executions of the protocol will not be able to detect any correlation among the values of the good players. A fortiori this will be valid for the players themselves and the adversary since we assume them to be polynomially bounded.

3. We present an attack on the Sako-Kilian election scheme presented at CRYPTO 94 [10]. The idea of using non-interactive proofs to reduce the rounds of communication in a protocol is not new. For example this is what Sako and Kilian do in their protocol. However one must be careful in the use of such a powerful tool. We will show that their protocol fails in achieving independence between casted votes. Using the techniques described in this paper we show how to modify their protocol in order to achieve this fundamental property.

2 Non-interactive zero-knowledge proofs

Before we dive into the description of our work, let us recall the notion of zero-knowledge proofs of knowledge without interaction. For details readers are referred to the original paper by Blum *et al.* [2] on non-interactive zero-knowledge proofs and the later one by De Santis and Persiano for the specific case of proofs of knowledge [4]. In this section we will also prove a technical lemma which will be useful to us later.

In a non-interactive proof, prover P and verifier V share a common input x and a random string σ. P runs on those input and produces a string π and V runs on x, σ, π for some polynomial time and either accept or rejects. We call $\sigma \circ \pi$ the *view* of the protocol.

A pair of Turing machines (P, V) constitutes a *non-interactive zero-knowledge proof system* (NIZKPS) for a language L if the following conditions are met

Completeness for all $x \in L$ and random strings σ, $V(x, \sigma, \pi)$ accepts with probability 1.

Soundness For all $x \notin L$, random strings σ and false proofs π' produced by any Turing machine P', $V(x, \sigma, \pi')$ rejects with high probability (i.e., $\geq 1 - 2^{2|x|}$)

Zero-knowledge There exists a probabilistic polynomial time Turing machine S called the *simulator* which on input $x \in L$ produces a view of the protocol with a probability distribution indistinguishable from the true one.

In [2] the authors describe a NIZKPS for the language SAT and so for all **NP**. In their protocol the Prover can be a polynomial time machine provided he knows a satisfying assignment for the formula. Protocols of this kind are called *proofs of knowledge*. In that case (see [4]) the soundness condition is changed to one requiring the Prover to know a witness of the theorem whenever he convinces the Verifier. By "knowing a witness" we mean as usual that there exists a knowledge extractor M which given access to P efficiently produces a witness.

In order to use non-interactive proofs in a distributed computing setting we have to make sure that a faulty process *really* gains nothing from seeing someone else's proof. For example if P_j publishes $s_j = E(b) \circ \pi$ where π is a NIZKP of knowledge of the bit b, a faulty P_i could post $s_i = s_j$ and pretend to have published (and to know) the same bit. In order to avoid that we are going to prove the following lemma:

Lemma 1 *Given a language L, and a NIZKPS (P, V) for L, n strings $x_1, \ldots, x_n \in L$, $n+1$ random strings $\sigma_1, \ldots, \sigma_n, \rho$, and $\pi_i = P(x_i, \sigma_i)$, if an algorithm $\mathcal{A}$ on input $x_i, \sigma_i, \pi_i, \rho$ outputs a string $y \in L$ and a π such that $V(y, \rho, \pi)$ accepts with non-negligible probability, then $\mathcal{A}$ knows a witness for $y \in L$, i.e., there exists an algorithm M that given access to $\mathcal{A}$ outputs a witness for $y \in L$ with non-negligible probability.*

What the lemma basically says is that if we change the random string then proofs relative to other random strings will be of no use to fabricate a new theorem and a "good proof" for it. The reason this is true relies on the simulatability of NIZKP.

Sketch of Proof If algorithm $\mathcal{A}$ comes up with the theorem and the proof efficiently given access to proofs produced by P, she could do that herself by running the simulator instead. So we would have an efficient algorithm convincing the Verifier with non-negligible probability and according to the soundness condition, $\mathcal{A}$ must know a witness of the theorem. ■

The lemma does not apply in the "copying" situation, because in that case the faulty processor is choosing the reference string as well as the theorem (both equal to the ones of the good processor) instead of picking it at random.

Notice that one of the consequences of this lemma is that access to the random string ρ before the choice of the theorem y, does *not* help in proving false theorems. This fact was already mentioned in the original paper [2] and will be important later.

2.1 The Fiat-Shamir heuristic

The Fiat-Shamir heuristic is a less rigorous but more efficient way of building non-interactive proofs. The heuristic is based on a secure hash function. When the prover wants to create a non-interactive proof of a theorem he just run the usual interactive protocol by himself. To do that honestly, he computes the challenges of the Verifier at each round by applying the hash function to his messages so far. This way he obtains random-looking bits and still the Verifier can check the process has been performed correctly. There is no proof that this method performs correctly, but in practice no attacks are known.

However also in this case we need some way to insure faulty processes cannot correlate their proofs to good processes ones. As before the idea is to make sure the random challenges are different, and the idea would be to "personalize" the messages that each party sends to the other one.

3 Simultaneous Broadcast

What exactly is *independence*? If we go back to the contract bidding example we notice that Harvard actions are not *independent* from MIT's ones because Harvard is acting based on MIT's messages.

In *simultaneous broadcast* we would like to have each player to behave as if the other players were not there or, in better words, regardless of what other players do during the protocol. If we model each player as a probabilistic Turing machines, then the value they announce is a function of their input and internal coin tosses and eventually of the announced values of the other ones. The key point in defining independence is to avoid this functional dependence by imposing that, say, P_i commits to a certain value with the same probability no matter what P_j has committed to. In this section we will try to cast this intuition in a formal definition.

In the following we will refer to a quantity as *negligible* if it can be made smaller than the inverse of any polynomial in a security parameter that we assume is given as common input to all the players.

A n-party protocol is a n-tuple of probabilistic polynomial-time interactive Turing machines (that we will call *players* in the following) $(P_1, \ldots, P_n)$. In our model these processors are connected by one-to-one communication channels and each processor has a dedicated broadcast channel. Channels are *public* i.e. any player can overhear the messages sent on any channel. We assume that there is an adversary $\mathcal{A}$ that decides *dynamically* which players corrupt and that coordinates the actions of the corrupted players. In our model the adversary is computationally bounded: indeed she can corrupt at most a fixed fraction of the processors and perform only polynomial time computations. The adversary is allowed *rushing* i.e. messages sent at round i by faulty processors may depend on messages sent at round i by the honest processors. With t we denote the *fault-tolerance* of the protocol, that is the number of players the adversary can corrupt.

Let's suppose without loss of generality that the value to be announced is a bit $b \in \{0, 1\}$. *Simultaneous broadcast* is a protocol composed by two parts `(Commit, Reveal)` that satisfy the following requirements.

Requirement 1: *Honest commitment*
At the end of `Commit` for each player P_i there is a fixed value $b_i \in \{0, 1, \star\}$ assigned to him. In other words there exist a computationally unbounded Turing machine that given as input a transcript of the protocol, outputs $b_1, \ldots, b_n$. If P_i follows the protocol then $b_i \in \{0, 1\}$. We will say that P_i *committed* to b_i

Given the previous requirement the following quantities are well defined: $\forall P_i$ $\forall r \in \{0, 1, \star\}^{n-1}$ $\forall b_i \in \{0, 1\}$

$$p^{P_i}_{b_i,r} = Prob[P_i \text{ commits to } b_i \text{ given that the other players are committing to } r]$$

When we say "given that the other players are committing to r" we mean that if r_j is the j-entry of the vector r then

- if $j < i$ then P_j is committing to r_j
- if $j \geq i$ then P_{j+1} is committing to r_j

We can now define the independence property as the second requirement that we ask from our protocol

Requirement 2: *Independence*
$\forall P'_1, \ldots, P'_n$ probabilistic polynomial time Turing machines, $\forall i$ if P'_i is good throughout the protocol then $\forall b_i \in \{0, 1\}$, $\forall r, s \in \{0, 1, \star\}^{n-1}$ the following quantity

$$|p^{P'_i}_{b_i,r} - p^{P'_i}_{b_i,s}|$$

is negligible.

This formalization guarantees statistical independence of the committed values. This is where our definition differs from the ones presented previously. In fact the definitions in [3, 5] require the values b_i and r to appear independent to a probabilistic polynomial time judge. If we assume the players to be computationally bounded this is enough to guarantee the independence of their actions.

We insist on statistical independence because (1) it is conceptually simpler, (2) it is a stronger requirement meaning that not only the players or the adversary, but also no way *observer*, no matter what her computational power might be, will be able to detect correlations, (3) it is not harder to achieve then "computational" independence.

Notice that we require independence only for the good players. Indeed since the adversary coordinates the actions of the corrupted players we cannot rule out the possibility of the values of bad players to be in some way related to each other. But with this definition we are sure at least that the value of a good player is independent with respect to those of *all* the other players, honest or corrupted.

Requirement 3: *Recovery*
At the end of the protocol each good processor P_i outputs an n bit vector $B_i = (B_{i,1}, \ldots, B_{i,n})$ such that

- if P_i and P_j are good processors then $B_i = B_j$
- the event that $\exists k$ such that $B_{i,k} \neq b_k$ happens with negligible probability

This requirement simply says that even if a player stops the protocol after having committed to a bit, it is still possible for the good players to recover his/her value with very high probability.

3.1 Previous solutions

As noted in the examples provided above, the main problem in achieving independence is that some parties may commit to a value without knowing it, but being sure that it is correlated to someone else's value. As pointed out in [3] and [5] the problem is eliminated by requiring each party to provide a zero-knowledge proof of knowledge of the committed bit. A simple solution would then be to have each player commit to his/her bit in a given order and then prove in zero-knowledge that he/she knows the value he/she has committed to. This solution however is very expensive in terms of round complexity since it requires $2n$ rounds of computation. Indeed to avoid correlation each proof must be conducted separately from the others and not concurrently. Suppose in fact that a faulty P_i and a correct P_j are concurrently providing zero-knowledge proofs, then when queried P_i could use P_j as an oracle to answer his queries.

Chor-Rabin in [3] solve this problem by a clever way of scheduling the zero-knowledge proofs. In their protocol each player P_i broadcast his encryption scheme

E_i and the encrypted value $E_i(b_i)$. Then each player proves in zero-knowledge that he knows b_i. To avoid correlation between the proofs this will be done $O(\log n)$ times in a way that for every pair of processors P_i, P_j there is a phase in which P_i acts as a prover and P_j only as a verifier. Moreover to allow recovery of the values they add a Verifiable Secret Sharing (VSS) of the value b_i. At the end this results in a $O(\log n)$ rounds protocol. The cryptographic assumption needed for the protocol is the existence of one-way functions.

3.2 Our constant rounds solution

Our solution to the simultaneous broadcast problem will be based on Chor-Rabin protocol. Each processor will broadcast his value encrypted and then a NIZK proof of knowledge of the value broadcasted. In doing this we will make sure that each player references to a different and independently chosen random string to prove his statement. Because of Lemma 1 this will eliminate the risk of correlation. Then each player will share his value among all other player using the VSS protocol from [8]. The total number of rounds will be constant. To be able to use NIZK proofs we must assume the existence of trapdoor permutations (see [6]).

We assume that the players share a random string σ nk bits long where k is the length needed from the reference string in order to do a NIZKP in our protocol. When P_i wants to produce a NIZKP (and as we will see he has to do that only once during the protocol) he will refer to the string $\sigma_i = \sigma[(i-1)k + 1 \ldots ik]$. We will show later how to eliminate this common randomness assumption.

In the following description of the protocol, let $t = \frac{n}{2}$ be the allowed fault tolerance.

Protocol 1

1. Each process P_i publishes his own public key encryption scheme E_i. E_i is actually a probabilistic encryption scheme as in [9]

2. Each process P_i publishes a string $s_i = E_i(b_i, r_i) \circ \pi_i$ where

   - b_i is the value P_i wants to announce
   - r_i are the random number used for the probabilistic encryption
   - π_i is a NIZK proof that P_i knows b_i relative to the random string σ_i

3. Each process checks on what P_i broadcasted, i.e. runs a verification procedure on the string s_i, and if the proof fails broadcasts a disqualification vote for P_i. If $t+1$ such votes are casted P_i is disqualified and his value is assumed to be $\star$

4. Each non-disqualified process P_i shares his value b_i among all players using the VSS protocol of [8]. I.e., P_i chooses a random polynomial $R_i(x)$ of degree t, such that $R(0) = b_i$. P_i sends to player P_j the value $R_i(j)$ encrypted with P_j's public key. Then he proves in zero-knowledge that the shared value is identical

to the one announced in round 1. This proof can be conducted interactively or to save rounds non-interactively as well.

5. Again each process checks on the VSS proof P_i performed during the previous round and if the proof fails, broadcasts a disqualification vote for P_i. If $t+1$ such votes are casted P_i is disqualified and his value is assumed to be $\star$

6. Each process P_i broadcasts the values b_i and r_i. If these values match the encryption broadcasted in round 1 all players accept b_i as P_i's announced value. Otherwise (or if P_i does not broadcast anything) the players run the recover phase of the VSS protocol and compute b_i on their own.

End of Protocol 1

The proof of the correctness and security of this protocol is quite simple. Intuitively the reasoning is as follows. Because of lemma 1 at the end of round 3 every player who has not been disqualified must know the bit he is committed to. This is the bit the player must share and eventually broadcast at the end. If a bad player manages to correlate his bit to the one of good player P_i, then, since he *knows* from the beginning the value of the bit, one can show that the encryption scheme E_i is not secure.

Theorem 1 *Protocol 1 is a constant round simultaneous broadcast protocol.*

Sketch of Proof The honest commitment and recovery requirements are easily seen to be met by the protocol. The main point is the independence requirement. We proceed by contradiction. Suppose that there exist an adversary $\mathcal{A}$ who is able to correlate i.e. $\exists$ a good player P_i and $\exists r, s \in \{0, 1, \star\}^{n-1}$ such that w.l.o.g.

$$|p_{0,r}^{P_i} - p_{0,s}^{P_i}| > \frac{1}{q(n)}$$

where q is a polynomial.

By an usual hybrid argument we can prove that $\exists t, u \in \{0, 1\star\}^{n-1}$ such that w.l.o.g.

$$t_i = u_i \forall i \neq j \text{ and } t_j = 0 \text{ and } u_j = 1$$

and

$$p_{0,t}^{P_i} - p_{0,u}^{P_i} > \frac{1}{p(n)}$$

with $p(n)$ polynomial

At this point we can construct an inverter I that is an efficient algorithm that breaks the encryption scheme E_i of player P_i using $\mathcal{A}$ as an oracle. On input $E_i(b_i)$, I runs a simulation of the first three rounds of the protocol. He plays the role of P_i on the network of players, using the advises of $\mathcal{A}$ to corrupt players. Under the condition that $\mathcal{A}$ does not corrupt P_i (this event must happen with non-negligible probability) we know that the bit of P_j is equal to the one of P_i with probability sensibly bigger than $\frac{1}{2}$. Running the knowledge extractor will return b_i with non-negligible probability. ■

A possible source of worry in this protocol is the fact that the random string σ_i is easily accessible to P_i before the beginning. But as we noticed in the previous section this will not help P_i in manufacturing a false proof. A way of eliminating the common randomness assumption would be to give to each player a string $\tau_i = \tau_{i1} \circ \tau_{i2}$ describing his identity (we can think of it as full name, date of birth, social security number etc.) known to everybody. Then assume the existence of a pseudo-random function generator like the one in [7]. Then $\sigma_i = f_{\tau_{i1}}(\tau_{i2})$.

Remark: We do not need to generate a new string σ_i for each time we perform the protocol. Indeed using results in [2, 6] it is known that multiple theorems can be proven using the same random string.

4 Election Protocols

At CRYPTO 94 Sako and Kilian presented a new voting scheme based on partially compatible homomorphisms [10]. Their scheme is based on a previous protocol of Benaloh and Yung [1]. Their improvements on the previous scheme are twofold:

- they use a more general family of homomorphic encryption functions based on a discrete-log like problem.
- they incorporate more modern techniques (which were not available at the time of the original paper of Benaloh and Yung) to improve the overall efficiency of the protocol.

In particular they make extended use of the Fiat-Shamir heuristic for removing interaction from proofs of knowledge and so improve substantially on the round complexity of the protocol.

In this section we will show that the scheme described in [10] fails on achieving independence between votes casted by different players. The problem lies on a wrong application of the Fiat-Shamir scheme that brings consequences similar to the ones described in Section 2.1. Using the techniques described in this paper we will also present various ways to fix these problems.

4.1 Sako-Kilian protocol

Let us summarize the Sako-Kilian protocol in its most important aspects. The protocol is based on a pair of *partially compatible homomorphic* encryption functions, i.e., a pair of functions $\{E_1, E_2\}$ over Z_q (with q prime) such that:

- $E_i(x + y) = E_i(x)E_i(y)$

- the two distributions :

  1. $(E_i(x), E_j(y))$ with x and y chosen uniformly
  2. $(E_i(x), E_j(x))$ with x chosen uniformly

  are computationally indistinguishable

Suppose there are two centers C_1, C_2 counting votes. Let k_1, k_2 be their public keys respectively for a fixed public-key encryption scheme E (which needs not to be an homomorphism). The protocol goes as following:

Vote casting Voter P_i chooses his vote v_i, 1 for a "yes" vote, -1 for a "no" vote. He then chooses randomly $x_{i,1}, x_{i,2}$ such that $v_i = x_{i,1}+x_{i,2}$. He posts $y_{i,1} = E_1(x_{i,1})$ and $y_{i,2} = E_2(x_{i,2})$ and proves in zero-knowledge that $x_{i,1} + x_{i,2} = 1$ or -1. We will describe this proof later. Then voter P_i posts $E(k_j, x_{i,j})$.

Vote counting Center C_j decrypts $x_{i,j}$ and checks that it agrees with $y_{i,j}$. Center C_j sums up all the $x_{i,j}$ to obtain t_j and posts t_j. Each voter checks that $E_j(t_j) = \Pi_i y_{i,j}$. Finally set $T = t_1 + t_2$. T is equal to the difference between "yes" and "no" votes.

The zero-knowledge protocol to check the correctness of the proof can be found in the original paper [10]. It is a straightforward commit-challenge-reveal protocol. To eliminate interaction the Fiat-Shamir heuristic is used. The protocol is run in parallel say 60 times (to achieve a probability of error $< 2^{-60}$) and the 60 bits of the verifier's challenge are obtained by applying a random-looking hash function to *the prover's first messages* (this is what Sako and Kilian suggest in their paper, we will see shortly how this is the cause of serious problems). Assuming that the hash function behaves as a random oracle then we can prove that the probability of cheating for the prover is $< 2^{-60}$. One of the main advantages of this approach is *universal verifiability* i.e., everybody can check the correctness of P_i's vote and of the entire election protocol.

With these improvements voter P_i must post just a single string s_i to cast his vote. s_i will be of the following form :

$$s_i = E_1(x_{i,1}) \circ E_2(x_{i,2}) \circ \pi_i \circ E(k_1, x_{i,1}) \circ E(k_2, x_{i,2})$$

where π_i is the non-interactive proof that the vote is correct. For ease of notation let us define

$$a_i = E_1(x_{i,1}) \circ E_2(x_{i,2}) \circ E(k_1, x_{i,1}) \circ E(k_2, x_{i,2})$$

i.e., s_i minus the proof π_i.

Suppose now voter P_j wants to copy P_i's vote. The only thing he has to do is to post the same exact message as P_i did. So P_j waits for P_i to cast his vote s_i and then posts $s_j = s_i$. To require every single voter to post a different string s is not satisfactory for two reasons:

1. it introduces interaction between the centers and the voters
2. it requires some extra assumption on the encryption functions E_1 and E_2 stating that given a string s it is infeasible to produce a string s' that cast the same vote

4.2 How to fix this problem

A first solution would be to construct differently the string π_i, that is the non-interactive proof that P_i's vote is correct. Instead of using the Fiat-Shamir heuristic we could directly use non-interactive zero-knowledge proofs. To do this we need shared randomness, i.e. we need to associate each voter with a publicly known random string σ_i. Alternatively we could use an identity string τ_i to generate a pseudo-random string σ_i as described in the previous section. Then π_i will be a NIZKP relative to the string σ_i. As we remarked before, advance knowledge of the string σ_i will not help P_i in fabricating a false proof. Such proof could be constructed by reducing the problem to a SAT formula and then using Blum *et al.* protocol. All these computations can be performed off-line, so the relative inefficiency of this method does not really constitute a serious problem.

A second, probably more efficient, solution can be obtained by modifying the implementation of the Fiat-Shamir heuristic in the following way. After running in parallel 60 copies of the first round of the interactive proof that the vote is a correct one, compute the challenges of the verifier as $h(a_i \circ \tau_i)$ where τ_i is the identity string of P_i. [2]

The first solution has the advantage of being provably secure and not an heuristic that holds conditioned to the "goodness" of the hash function.

5 Conclusion

We have put forward a new and stronger formal definition for the problem of independence in distributed computation. A new constant round protocol for the task of simultaneous broadcast has been presented. Previous solutions were all $O(\log n)$ rounds. Moreover as a practical application of this problem we have presented some critical remarks on the security of the Sako-Kilian voting scheme. The remarks stem from the failure of that protocol in achieving independence between casted votes. In the spirit of our results we have finally proposed some simple modifications to the Sako-Kilian protocol which greatly enhance its security.

[2] After being notified by the author of this paper of the problem with their scheme, Sako and Kilian independently found a similar fix to this one

Acknowledgments

We wish to thank Silvio Micali, Ray Sidney and Joe Kilian for useful discussions and suggestions.

References

[1] Josh Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *5th ACM Symposium on Principles of Distributed Computing*, pages 52–62, 1986.

[2] Manuel Blum, Alfredo DeSantis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, December 1991.

[3] Benny Chor and Michael Rabin. Achieving independence in logarithmic number of rounds. In *6th ACM Symposium on Principles of Distributed Computing*, pages 260–268, 1987.

[4] Alfredo DeSantis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction. In *33rd IEEE Symposium on Foundations of Computer Science*, 1992.

[5] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[6] Uri Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero-knowledge proofs. In *31st IEEE Symposium on Foundations of Computer Science*, 1990.

[7] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33:792–807, 1986.

[8] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[9] Shafi Goldwasser and Silvio Micali. Probabilisitc encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[10] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In *CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.