Achieving independence efficiently and securely

Rosario Gennaro
Laboratory for Computer Science
Massachusetts Institute of Technology *

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 of knowledge [3]. To explain the concept of independence let us present some motivating scenarios first.

Example 1 *Contract bidding* : Town T decides to sell part of their estate 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 companies A and B are interested in buying the estate. A places its bid b_A by sending $m_A = E(b_A)$ over an insecure phone line. B is tapping the line and overhears A's bid. In spite of the security of the encryption scheme E, now B is able to at least tie A's bid, by sending over $m_B = m_A$. Even requiring B to send a different cyphertext would not solve the problem since it could still be possible to produce a cyphertext $m_B \neq m_A$ 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 not to 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 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.

1.1 Previous work and our contribution

Chor and Rabin in [1] 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. In this paper we improve on this result. More in detail:

(1) We describe a *constant* round protocol for simultaneous broadcast under a general cryptographic assumption. In doing that we will heavily rely on the properties of non-interactive zero-knowledge proofs of knowledge (see [3].)

(2) We refine the definition of independence to a stronger one. In fact in [1] 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.

2 Simultaneous Broadcast

What exactly is *independence*? If we go back to the contract bidding example we notice that B's actions are not *independent* from A's ones because B is acting based on A'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 latter functional dependence by imposing that, say, P_i commits to a certain value with

*Postal address: 545 Technology Square, Cambridge MA 02139, USA. Email address: rosario@theory.lcs.mit.edu. Research supported by NSF grant no.9121466-CCR

Protocol Sim-Br

1. Each process P_i publishes his own public key encryption scheme E_i. E_i is actually a probabilistic encryption scheme.
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 NIZKPK 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 [2]. 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.

Figure 1: A constant round protocol for simultaneous broadcast

the same probability, no matter what P_j has committed to. The formal definition is the following:

Let [1]

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

Independence:
$\forall P'_1, \ldots, P'_n$ player, $\forall i$ 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 [1] 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.

2.1 Our constant rounds solution

Each processor will broadcast his value encrypted and then a *non-interactive zero-knowledge proof of knowledge* (NIZKPK) 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. This will eliminate the risk of correlation. Then each player will share his value among all other player using a Verifiable Secret Sharing (VSS) protocol like the one in [2]. The total number of rounds will be constant. To be able to use NIZKPK we must assume the existence of *Dense Secure Public-Key Cryptosystems* (see [3].)

We assume that the players share a random string σ of length nk where k is the security parameter (and so we can assume that in order to produce a NIZKPK in our protocol one needs a reference string of length k.) When P_i wants to produce a NIZKPK he will refer to the string $\sigma_i = \sigma[(i-1)k+1 \ldots ik]$. The full protocol appears in Figure 1.

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

Intuitively the reasoning is as follows. 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.

[1] 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 else if $j \geq i$ then P_{j+1} is committing to r_j

References

[1] 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.

[2] 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.

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