Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults

(Extended Abstract)

Benny Chor(1) Shafi Goldwasser(2) Silvio Micali(3) Baruch Awerbuch(4)

Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technlogy Square, Cambridge MA 02139

1. Introduction

This paper contributes two new primitives: *verifiable secret sharing* in the area of cryptographic protocols and *simultaneous broadcast* in the field of fault tolerant computation. Many problems, such as distributed coin flipping and "uninfluenced" voting, can be reduced to the solution of our primitive problems.

Verifiable secret sharing is a cryptographic protocol that allows one to break a secret in n pieces and publicly distribute them to n people so that the secret is reconstructible given only sufficiently many pieces. The novelty is that every one can verify that all received a "valid" piece of the secret without having any idea of what the secret is. One application of this tool is the simulation of simultaneous-broadcast networks on semi-synchronous broadcast networks.

Simultaneous-broadcast networks are a powerful kind of communication networks. Informally, they guarantee that all processors send messages simultaneously (at designated times) and all processors receive messages simultaneously (at designated times). Designing distributed protocols robust against Byzantine faults (as well as proving their correctness) is extremely easy for such networks.

Under the assumption that it is infeasible to invert the RSA function, we show how to construct a simultaneous-broadcast network with n processors and t Byzantine faults from a semi-synchronous network with n processors and t Byzantine faults. Our construction actually consists of a *compiler* that transforms algorithms, robust against t Byzantine failures in a simultaneous network, to equivalent algorithms robust against t Byzantine failures in a semi-synchronous network.

The transformed algorithms require $O(1)$ rounds of communication for every original round and $O(2^t)$ local computation time. Thus, it is fully polynomial in n when $t=O(\log n)$. Similar results are proved for $t=\log\log n$ under the assumption that factoring is intractable.

2. Simultaneous Broadcast

The idea of a simultaneous broadcast network is best illustrated by the following real life example.

Example: Let $A_1,..., A_n$ denote n people in a room.

(1) Research supported by IBM Graduate Fellowship and Bantrell Postdoctoral Fellowship.
(2) Research Supported by NSF Grant DCR-8509905.
(3) Research Supported by NSF Grant DCR-8413577.
(4) Research Supported by Weitzmann Postdoctoral Fellowship.

Each A_i has his own blackboard and piece of chalk. No one can erase or write on others boards. The room they are in is designed so that it is lighted for one hour, dark for the next, then lighted again for an hour, dark for the next, and so on. $A_1,..., A_n$ communicate as follows. When the room is dark, each one writes a message on his board. No one can read the message till the light comes on. When the light switches on, everyone *simultaneously* reads everyone elses blackboard. The j-th message of A_i is defined to be what appears on A_i's black board at the beginning of the j-th "lighted" interval. Whatever is written when the light is on is ignored.

2.1 Intuition

The above intuitive model of communication possesses three interesting properties. These properties hold even if some people decide to secretly talk to one another during the darkness interval, in order to decide what to put on their boards.

Informally,

1) If A_i does not collaborate with anyone upto the j-th round, everybody's j-th message is "independent" of A_i's j-th message.
2) No message can be "erased" or altered after the light switch goes on. (If A_i's board is empty, he has delivered the empty message).
3) All of $A_1,..., A_n$ agree on what the j-th message of everyone is.

The first property captures simultaneity. The last two capture broadcast.

Let us now formalize all of the above in the notion of ideal simultaneous broadcast.

2.2 Broadcast and Simultaneous Broadcast Networks

Let NET denote a communication network whose processors $A_1,..., A_n$ communicate by sending and receiving messages over a common communication channel.

In addition to sending messages to one another the processors perform internal computations (toss coins, read their work tapes, etc).

With respect to an execution of a given protocol $PROT$, a processor A_p is *proper* if it always computes according to $PROT$. A processor which is not proper is called *faulty*. Faulty processors can coordinate their strategies to try to disrupt the system. This is done as follows. When an execution of a protocol $PROT$ is started, an execution of a special algorithm, the *adversary*, also begins. The adversary chooses which processors to make faulty in a dynamic fashion during the execution of $PROT$.

Once the adversary chooses a processor to be faulty, it takes control over the messages that processor sends. Let us stress that

all protocols and algorithms considered in this paper are probabilistic.

Definition 1 (Broadcast Networks): A communication network is a *broadcast network* if

1) a unique message (string) m_{ij} is associated with each processor A_i and each positive integer j.

   m_{ij} is called the j-th round message of A_i.

   Let $H_j = \{(i,j,m_{i,x}) : 1 \leq i \leq n, 1 \leq x \leq j\}$ be the history of communication up to and including round j.

A broadcast network is an *n-t-broadcast-network* if for all protocols PROT and all adversaries ADV, the following holds:

2) ADV makes at most t processors faulty during an execution of $PROT$.

   The set of processors that are chosen to be faulty at round $j+1$, F_{j+1}, is computed by ADV on inputs H_j and $\bigcup_{x=1}^{j} F_j$.
3) A proper processor A_p computes $m_{p\,j+1}$ by running $PROT$ on inputs p and H_j.
4) The round j messages of the faulty processors are computed by ADV on inputs H_j and the messages of the proper processors at round $j+1$.

Let the algorithms $PROT$ and ADV in a n-t-ISB network NET be given. Then, $H_j(NET,PROT,ADV)$ will denote the conditional probability distribution on H_j given H_{j-1}. The probability is taken over all coin tosses of PROT and ADV.

Definition 2 (ISB networks): A $n-t$-broadcast network is *n-t-ideal simultaneous network* if the following more restrictive condition 4′ hold for the messages sent by the faulty processors instead of of condition 4:

4') The round j messages of the faulty processors are computed by ADV only on input H_j.

(Comment: if in definition 2, j is interpreted as time, then all messages of round j cannot depend on current messages of proper processors. The only dependence allowed is on their past messages.)

2.3 The Power of ISB Networks

Simultaneous broadcast networks are a fundamental primitive as simultaneity lies at the heart of many protocols: coin flipping, fair voting, contract signing, exchanging secrets etc.

All these protocols are in fact extremely easy to implement in an simultaneous broadcast network. Let us consider two examples.

Coin flipping: All proper processors should agree on a randomly selected bit. In a simultaneous broadcast network, it is enough that at common time i all processors broadcast a randomly chosen bit. At time $i+1$, all proper processors will take the XOR of the broadcasted bits as the outcome of the coin toss. (Remember that all processors are forced to broadcast. If they broadcast something different from 0 or 1, their bit will be considered to be 0 by default). The outcome of the coin toss will be the same because of property (1) of a broadcast Network. It is enough that a single proper processor belongs to the network for the outcome to be unbiased.

Uninfluenced voting: Informally, each processor should vote for its "best" candidate, uninfluenced by the vote of others. (For example, there may be some tendency to vote for the eventual winner!). This is easily achieved by having all processors vote at time i and take the majority at time $i+1$.

2.4 Simulation of ISB Networks

Though very powerful, ISB networks appear to be quite an idealized model of communication. One contribution of this paper is showing that ISB networks can be simulated on broadcast networks. The proposed simulation necessarily only "approximates" the mechanism of an ISB, but captures all of its properties with respect to poly time computation.

Our simulation of an ISB network is a virtual one. Roughly, the processors in the broadcast network will actually exchange auxiliary messages for which property 4' of a ISB network does not hold. The messages $\{m'_{ij}\}$, that satisfy "all" properties of messages sent in an ISB network, are never sent but are computed from the auxiliary messages. Let us formalize this notion of simulation.

Definition 3 (compiler): A *compiler* is a pair of algorithms (C,D). C is called the *coding algorithm* and D the *decoding* algorithm.

Let PROT be a protocol defined for an n-t-ISB network. On input (a description of) PROT, C computes (a description of) a protocol $PROT_C$ defined for an n-t broadcast network NET'.

Let the proper processors of an n-t broadcast network execute the protocol $PROT_C$ in presence of an adversary ADV'. Let h_j be the set of all messages sent from round $c(j-1)+1$ to round $c \cdot j$ during such an execution. Algorithm D, on input h_j and integer $1 \leq i \leq n$ computes a string m'_{ij}. (Here c is a constant determined by the compiler only).

Let $H'_j = \{(i,x,m'_{i,x}) : 1 \leq i \leq n, 1 \leq x \leq j-1\}$. For given compiler (C,D), protocol $PROT$ and adversary ADV', let $H'_j(NET',C,D,PROT,ADV')$ denote the conditional probability distribution on H'_j given H'_{j-1}.

Definition 4 (simulator): Let NET' be an n-t-broadcast network and NET be an n-t-ISB network. An *n-t-simulator* is an efficient compiler (C,D) such that for all efficient protocols $PROT$ defined for NET, and for all efficient adversaries ADV' defined for NET', there is an efficient adversary ADV for

NET such that, for all $j>0$ $H'_j(NET',C,D,PROT,ADV')$and $H_j(NET,PROT,ADV)$ are polynomial-time indistinguishable.

(Here efficient means running in probabilistic polynomial-time in some common security parameter K, presented as input to all processors).

The notion of polynomial-time indistinguishable probability distributions is that of [GMR] (which is derived from [GM] and [Y1]). For convenience, this definition is recalled in the next subsection.

Comment: Intuitively, definition 4 says that whatever the faulty processors can achieve in a broadcast network simulating a ISB network, can also achieve in a true ISB network. In other words, if the messages produced by a protocol satisfy certain properties in an ISB network the messages "sent" in the simulation of the protocol on a broadcast network satisfy the same properties (with respect to polynomial-time observers). Therefore "correct" protocols get mapped to "correct" protocols.

2.4.1 Indistinguishability of probability distributions

One more technical definition- that of polynomial indistinguishability (taken from [GM], [Y1],[GMR]) is needed in order to describe our results.

Let $I \subseteq Z^+$ be an infinite index set (of the security parameter) and let c a positive constant. For each $K \in I$, let Δ_K be a probability distribution over the K^c long binary strings (messages). We call $\Delta = \{\Delta_K \mid K \in I\}$ an *I-c-ensemble.* By saying that Δ is an *ensemble* or an *I-ensemble* we mean, respectively, that there exist I and c or simply c such that Δ is a I-c-ensemble.

Informally, we say that two ensembles (or probability distributions) are indistinguishable if no polynomial-time computation can tell them apart. Formally,

A *distinguisher* is a probabilistic polynomial-time algorithm D that on input a string b outputs a bit b. Let $\Delta_2 = \{\Delta_{2,K} \mid K \in I\}$ and $\Delta_1 = \{\Delta_{1,K} \mid K \in I\}$ be two I-c-ensembles. Let $p^D_{K,1}$ denote the probability that D outputs 1 on a K^c-bit long input string randomly selected with probability distribution $\Delta_{2,K}$. Symmetrically, $p^D_{K,2}$ denotes the probability that D outputs 1 on a K^c-bit long input string randomly selected with probability distribution $\Delta_{1,K}$.

Definition 5: We say that the ensembles Δ_2 and Δ_1 are *indistinguishable* if for all distinguishers D, $|p^D_{K,1} - p^D_{K,2}| < \frac{1}{K^d}$ for all d and sufficiently long K.

2.5 Our result

Theorem: Under the intractability assumption for the RSA function (section 3.4), we exhibit an $n - O(logn)$ simulator.

More generally, under the assumption that for large enough K it is infeasible to invert RSA on moduli product of two K bit long primes, we show how to perform the following simulation:

> A protocol for simultaneous broadcast networks with n processors and t faults ($t<4n$) is simulated on a broadcast network with n processors and t faults ($t<4n$). The simulation overhead is $O(1)$ in communication rounds and $O(2^t + K)$ in local computation time.

While in general fully polynomial time algorithms are preferable, there are some important cases where $O(2^t)$ processing time is acceptable. One such example is Bracha's randomized Byzantine agreement algorithm [Br], where n denotes the size of a subcommittee. These subcommittes are logarithmic in the size of the whole network, and so (single) exponent in the size of the subcommittee is still polynomial in the size of the whole network.

2.6 Discussion

Simultaneous Broadcast versus Byzantine Agreement

Tollerating O(log n) faulty processors is an almost trivial task with respect to the problem of achieving Byzantine Agreement on a point-to-point synchronous network. This may lead us to the belief that tollerating $O(log\ n)$ faulty processors with respect to achieving simultaneous broadcast on a semi-synchronous broadcast network is equally trivial. However, achieving simultaneity appears to be a much harder problem than achieving Byzantine agreement. This is best illustrated by the following "flawed" attempt to reach simultaneous broadcast on a synchronous broadcast network.

A naive and flawed attempt:

To send the j-th message m_{ij}, processor A_i of a broadcast network sends the encryption of m_{ij} at round $2j+1$. All proper processors read all such ciphertexts. If a processor sends nothing or two or more ciphertexts, a fact noted by all processors, then its m_{ij} is considered to be the empty message. At round $2j+2$ all processors should send the decryption of their messages. Failure of sending this decryption is again interpreted as the empty message. (We are assuming that the encryption scheme is such that there is a unique possible decoding for each ciphertext. Moreover, for this example, we assume that, on input x and y, one may decide whether "y is the encoding of x").

Notice that *the above procedure does not make the network simultaneous.* In fact, before round $2j+3$, all faulty processors know what the proper processors' messages are. By either sending or not sending the decryption of its message, a faulty processor has an option of sending its originally chosen message or the empty one. As it can make this decision after having seen the other processors' messages, this violates the "simultaneity" condition. Just see how disruptive this could be in the coin flipping protocol above described. Once a faulty processor sees the decryptions of the bits of all other processors he can choose whether to send the decryption of his own bit or not. In one case the network chooses the XOR of all bits as the outcome of the coin toss, in the second one the XOR of "0" and all other bits.

2.7 Implications of the main result

Given a "compiler" for simultaneous-broadcast networks, protocols for coin flipping and fair voting can immediately and correctly be implemented on a semi-synchronous broadcast network (or even weaker type of networks) with O(log n) faulty processors. These implementations are obtained by "compiling" the simple protocols outlined above for simultaneous broadcast network. As for any general simulation, some of these compiled solutions can be more efficiently solved by ad hoc algorithms. For example, coin flipping is as rigorously but more efficiently solved in [Y2] and [ABCGM]. However, some other problems receive their first solution through our general compiler, as is the case for fair voting. For both coin flipping and fair voting, what is striking is the extreme simplicity of the original protocols. This exemplifies the power of simultaneous broadcast as a new primitive.

Our simulation uses as a subroutine the new tool of Verifiable Secret Sharing (VSS). In the next section, we describe the concept and implementation of VSS.

3. Verifiable Secret Sharing

3.1 Properties

In [S], Shamir proposed a method to share a secret that is briefly describe below. A *dealer* wants to break a secret b into n pieces $b_1, \ldots, b_n$ so that b cannot be reconstructed from any collection of t pieces ($t<n$), but it can be easily reconstructed from any collection of $t+1$ pieces. The dealer does so by choosing at random a polynomial $P(x)$, of degree t over the field Z_p (for a suitably large prime p), so that b is the free term of $P(x)$. Then $b_1=P(1), \cdots, b_n=P(n)$ are the desired pieces.

Shamir's scheme is secure in an information theoretic sense. However, this is based on having a non-faulty dealer with the ability to distribute pieces *privately.* In the context of a distributed network with faulty processors, it is not possible to send pieces in cleartext over the public network. If a dealer still wishes to distribute the pieces, he can send s_i encrypted under the recipient public key - $E_i(b_i)$. However, at this point the scheme is at most as secure as the encryption function E. It is even conceivable that E is secure, but s can be reconstructed from $E_1(b_1)$,..., $E_n(s_n)$. A more severe difficulty is the following:

Assume that the dealer is dishonest. Then, given one piece b_i, there is no way to *verify* that indeed b_i is a "valid" piece of the secret. In fact, the owner of b_i can never be sure that the polynomial interpolating the points not in his possession (and which he does not know) indeed passes through his own point $(i,P(i))$. Even worse (from some point of view), *there is no secret to speak of.* I.e. there is no specification of the secret given in advance.

These difficulties make Shamir's secret sharing hard to use in a fault tolerant distributed environment.

We provide an different method to share a secret. Typically, the secret is a single bit b that is chosen equal 1 with some a priori probability r (without loss of generality, $r=1/2$). This type of secret is very general since by concatenation it allows to have longer strings with arbitrary probability distributions.

• The security achieved by our protocol is that it is *infeasible* for less than t participants to cooperate and distinguish b from a random bit. More formally, no probabilistic polynomial time algorithm that monitors the public communication in the entire network and has access to the internal state of any t participants can output b correctly with probability greater than $\frac{1}{2}+\frac{1}{K^c}$ for all constants c and sufficiently large K.

• On the other hand, $n-t$ participants can reconstruct b, from their shares, with very high probability ($>1-\frac{1}{2^{n-3t-1}}$). Thus the t faulty processors cannot find b by themselves, but the $n-t$ correct processors can, regardless of the faulty processors behavior.

• The distribution of pieces is done *publicly*, in front of all participants. First, the dealer establishes the secret b by announcing a public encryption key E and an encryption $E(b)$ of the secret bit (where E is a RSA based probabilistic encryption scheme [GM],[ACGS]). Then all participants, one by one, play an interactive protocol with the dealer over the network channel to get their "share" of b.

• The effect of this protocol is that each participant C gets one *verifiable* piece b_C of A's secret: Every other participant D can verify that the piece C got is indeed a legitimate piece of the secret.

3.2 Oblivious Transfer

A crucial component of our verifiable secret sharing is a generalization of the oblivious transfer protocol. The beautiful notion of oblivious transfer (OT) was introduced by Rabin [R2]. It involves two parties A and B and an integer N_A (product of 2 large distinct primes) whose factorization is known only to A. A and B would like to play a protocol at the end of which

1) B is able to output N_A's factorization with probability essentially equal to 1/2 and
2) A does not know whether or not B has received N_A's factorization.

Rabin originally suggested an implementation of OT. Even, Goldreich and Lempel [EGL] generalized it. Fischer, Micali and Rackoff [FMR] found an interactive protocol (four rounds of communication exchange) which achieves the above properties (1) and (2) and is its correctness is provably equivalent to factoring.

In this paper, we use a generalization of the [FMR] protocol to moduli N_A's that are product of l primes. In this case the desired properties are

1) B gets exactly one split $N_A=S_1 \cdot S_2$ with uniform probability distribution among all possible splits (including the empty split $N_A=N_A \cdot 1$).
2) A should not know what split B actually received with probability greater than $\frac{1}{2^l}$.

Oblivious Transfer between A and B on integer N_A

Part1: B sends A a square modulo N_A, u_A^2 together with a 0-knowledge proof that B knows one square root of u_A^2 mod N_A. (for details see [FMR] or [GMR]).

Part2: A gives B a random square root of u_A^2 mod N_A.

Four rounds of message exchanges are required in the implementation. The effect of part 1 is that A is convinced with probability greater than $1-\frac{1}{2^K}$ that B knows one square root of u_A^2 but A gets no information as to which root it is. By saying "B knows a square root" we mean that B can computed such square root by running B_i twice on the same input and same coin tosses.

In the correctness proof for our VSS protocol, we regard the oblivious transfer axiomatically, as a protocol with the specified properties.

3.3 Our Verifiable Secret Sharing (VSS) Protocol

Code for dealer A
to verifiably share the secret bit b:
(for n participants, t faults, security parameter K)

Let $l=2^t+1$.

Part I (Preparations).

1. Pick l primes $p_1,\ldots,p_l$ of length K bits each.
2. Compute $N_A=p_1\cdots p_l$.
3. Pick $x_A \in Z_{N_A}^*$ uniformly at random such that $x_A \bmod N_A$ has least significant b.
4. Compute $y_A=x_A^3 \bmod N_A$.
5. Broadcast N_A and y_A.
 {at this point A is committed to b.}

Part II (Distribution of Pieces).

6. Give every other participant C a *random* split of N_A by running the oblivious transfer between A and C on N_A.

==========================

Turning Point

==========================

Part III (Reconstruction of Secret).

7. Each other participant C broadcasts the split of N_A that C holds.
8. If C detects that A deviated from the protocol before the turning point, C takes the default $b=$"empty message". Otherwise, C takes all (at least $n-t$) splits of N_A, and tries to factor N_A completely. If N_A does factor, has exactly l prime factors, and the exponent 3 is relatively prime to $\varphi(N_A)$, then C computes $x_A=\sqrt[3]{y_A} \bmod N_A$. C takes the least significant bit of x_A to be A's secret, b. If any of the above conditions do not hold, C takes the default $b=$"empty message".

3.4 Proof of Correctness

We prove the following statements:

1) If A deviated from the protocol before the turning point, then all non-faulty processors will take the default $b=$"empty message". If A followed the protocol till the turning point (steps 1--5) then, after the turning point (at step 8), all non-faulty processors will compute b (with overwhelming probability). This is proved in Lemma 3.

2) If RSA inversion is infeasible, then the probability that the faulty processors can predict b before the Turning Point is less than $\frac{1}{2}+\frac{1}{poly(K)}$ for any polynomial P and sufficiently large K. This is proved in theorem 5.

Let $N=p_1\cdots p_l$. Then we call the pair of integers (c,d) a *split* of N if $N=c\cdot d$. Let $splits(N)=\{(c,d) \mid N=c\cdot d\}$

Fact 1: Let N_A be as specified in the above protocol. On input pair (u,v) such that $u^2 \equiv v^2$ mod N_A and v was randomly picked among all the roots of u^2 mod N_A, it is easy to compute a random split of N_A.
Proof: A simple extension of Rabin's digital signature scheme argument.

Definition 5: Let N_A be as specified in the above protocol, and let $M=p_1p_2$ divide N_A. We say that M is *unseparated* by a set of t processors $B_1,\ldots,B_t$ if for each of the t splits $N_A=S_1^iS_2^i$ that B_i gets by the oblivious transfer ($i=1,\ldots,t$), either M divide S_1^i or M divide S_2^i.

Notice that the notion of unseparated M

depends only on splits which are received via the oblivious transfer, and is thus well defined even if factoring is easy.

Fact 2: If A is a correct processor, then for any t processors there is at least one pair of primes $p_1 p_2$ which divide N_A, such that M is unseparated by that set.

Proof: N_A has 2^t+1 prime factors, and every split partitions them to two subsets. Different partitions correspond to the subsets formed by intersections. Since t partitions can at most yield 2^t subsets, there must be at least one pair of primes, p_1, p_2 which remains in the same subset. Thus there is an $M=p_1 \cdot p_2$ unseparated by $B_1, \ldots, B_t$ at the turning point. **QED**

Lemma 3: Let N_A be a product of l distinct primes. Suppose processor A behave according to the protocol upto the turning point. Then at step 8, the $n-t$ non-faulty processors can factor N_A with probability greater than $1 - \frac{1}{2^{n+1-3t}}$.

Proof: The non-faulty processors at step 8 have at least $n-t$ random splits of N_A. We can represent every split by a 0 - 1 coloring of the l primes: Two primes are colored the same according to a given split if they are in the same factor of N_A according to that split. What is the probability that there exists a pair of primes p_i, p_j which are kept unseparate by all $n-t$ splits? To answer this question, consider a $(n-t)$-by-l matrix with zero-one entries. The l-th row correspond to the l-th split, and the columns i,j have the same entry (either both 0 or both 1) iff the primes p_i, p_j were not separated by this split.

Since each row correspond to a random split, the matrix is a random zero one matrix. The probability that N_A is not totally split equals the probability that there are two identical columns in the matrix. But

$$Prob(\text{ column}_i = \text{column}_j\ ) = \frac{1}{2^{n-t}}$$

Hence

$$Pr(\exists\ i,j \text{ with column}_i = \text{column}_j) < \frac{l^2}{2 \cdot 2^{n-t}}$$

Substituting l by 2^t+1, this probability is bounded above by $\frac{2^{3t}}{2^{n+1}}$. Hence with probability greater than $1-\frac{2^{3t}}{2^{n+1}}$ there exists no such pair i,j. In this case the non-faulty processors can compute the complete prime factorization of N_A, using at most $O(l(n-t))$ arithmetic operations. **QED**

Before proving the unpredictability of the secret at the Turning Point , we state exactly the RSA intractability assumption, and give two claims that are used in the proof.

RSA Intractability Assumption (RIA)

Let F be a probabilistic polynomial algorithm that takes as input an integer M, product of two K bit primes, both congruent to 2 mod 3. Let $\alpha_F(K)$ be the fraction of such moduli M on which F inverts RSA (x^3 mod N). Then, for all polynomial time algorithms F, for all $c > 0$, and for all sufficiently large K, $\alpha_F(K) < \frac{1}{K^c}$.

Remark: As stated, the assumption is quite strong. First, it corresponds to RSA with the specific power 3. Second, it requires that RSA be invertible on a subpolynomial fraction of instances only. It is possible to relax this assumption by replacing the last sentence by "$\exists\, c > 0$ such that $\alpha_F(K) < 1-\frac{1}{K^c}$". Using Yao's exclusive-or construction [Y1], we can modify our protocol to be proved correct under this weaker assumption. Also the power 3 can be generalized. For this abstract, we chose to give the simpler version of the protocol and make a stronger assumption.

Our next result, Theorem 4, states that the bit security of RSA over multi-prime modulus is not compromised even if almost all the factorization of the modulus is known.

Theorem 4: The following three problems are computationally equivalent (each is probabilistic poly(K)

time reducible to the other).

(1) Invert RSA ($x^e \bmod M$) on the modulus $M=p_1p_2$, product of two K-bit long primes where $(e,\varphi(M))=1$.

(2) Given $M=p_1p_2$ (product of two K-bit long primes), e such that $(e,\varphi(M))=1$ and $x^e \bmod M$, guess the least significant bit of x with success probability greater than $\frac{1}{2}+\frac{1}{poly(K)}$.

(3) Given $N=Mp_3...p_l$, e such that $(e,\varphi(N))=1$, $x^e \bmod N$ and $M,p_3,...,p_l$ (where $M=p_1p_2 \geq 2^l$ and all primes p_i are K bits long), guess the least significant bit of x with success probability greater than $\frac{1}{2}+\frac{1}{poly(K)}$.

Proof: The proof is a simple extension of the bit security result for two prime moduli [ACGS], and is omitted from this abstract. Details can be found in [C]. **QED**

The reader might be tempted to think that fact 2 together with theorem 4 imply the unpredictability of b at the Turning Point. Such impression is wrong as we still have to show that the adversary essentially cannot "gain any additional knowledge" in the course of the protocol execution of part 2 (the distribution of the t splits of N_A).

The success probability of an adversary in determining b is averaged over all A's choices of primes $p_1,..,p_l$ and over all executions of the verifiable secret sharing protocol (given a specific choice of primes). In the proof, it will be more convenient to think of the primes as fixed. To that end, suppose the adversary can guess b with probability $\frac{1}{2}+\varepsilon$. From fact 2, we know that at least one pair of primes $M=p_1p_2$ is unseparated at the Turning Point. Let S_{m,N_A} be the conditional success probability of the adversary in guessing b, given that A's modulus is N_A, M divide N_A and that M is unseparated at the Turning Point. The following fact is proved by a simple counting arguments.

Fact 5: For at least ε fraction of of all M and N_A (where N_A, M are as above), $S_{m,N_A} > \frac{1}{2} + \frac{\varepsilon}{4}$.

We will call M and N_A for which $S_{m,N_A} > \frac{1}{2} + \frac{\varepsilon}{4}$ *lucky number* and *lucky extension*, respectively.

In the proof of the unpredictability theorem, F will denote a probabilistic polynomial time algorithm which tries to predict b. $B_1,...,B_t$ will denote the probabilistic polynomial time algorithms corresponding to t faulty processors. F has access to the internal state of these t B_i-s (i.e can read their worktapes, change the coin tosses they use etc).

Theorem 5 (Unpredictability Theorem): Suppose A is a correct processor who chose his secret bit b with a priori probability 1/2. Then, under RIA, no polynomial time adversary F controlling any collection of t processors can predict b before the turning point, with success probability greater than $\frac{1}{2} + \frac{1}{poly(K)}$ (for all polynomials *poly* and all sufficiently large K).

Proof Sketch: The high level plan of the proof is to show that if an adversary F can predict b with success probability $\frac{1}{2}+\varepsilon$ then he also can, by simulating instances of the protocol, invert RSA, on ε fraction of moduli M's in expected time $O(1/\varepsilon)$. Thus from RIA ε must be smaller than any polynomial fraction $1/poly(K)$.

We now specify a guessing algorithm G. G takes as input a lucky composite number M product of two K-bit prime factors, both congruent to 2 mod 3, and a random cube $y=x^3 \bmod M$. G goal is to guesses the least significant bit of x with success probability $>\frac{1}{2}+\frac{\varepsilon}{4}$. G will use as subroutines the algorithms $B_1,...,B_t$ of the faulty processors, and the adversary prediction algorithm F.

G starts by picking at random $l-2$ primes $p_3,...,p_l$ each K bits long and congruent to 2 mod 3. Then, G computes $N_A=M\cdot p_3p_2\cdots p_l$, and a random y_A in the range $0<y_A<N_A$ satisfying $y_A \equiv y \bmod M$. (Notice that with probability at least ε, N_A is a lucky extension of M.)

By now, G has simulated steps 1--5 of the verifiable secret sharing protocol. He now simulates step 5, by broadcasting N_A and y_A. To simulate step 6 of the protocol, G must simulate A's role in $n-1$ oblivious transfers on the composite N_A.

How can G do this? Let's concentrate on Gs interaction with the faulty processors. In the real OT protocol A knew all factors of N_A and thus during the course of the OT protocol, could give each of $B_1,\ldots, B_t$ a random split of N_A. G, on the other hand, knows all factors of N_A except those which split M and thus can't give B_i a random split of N_A. But what it can and will do is simulate runs of the OT in which A gives B_i a split of N_A picked at random among all those splits which do not separate M. The interaction between G and $B_1,\ldots,B_t$ goes a follows.

> First $B_1,\ldots, B_t$ are supposed to choose elements $u_1,\ldots,u_t$ in Z_N^*. (They may do this jointly in any fashion: randomly or otherwise but in polynomial time.)
>
> Second, they transmit to G (simulating A), u_1^2 mod $N_A,\ldots,u_t^2$ mod N_A and "interactive proofs" (as defined by the OT protocol) of the fact that they "know" $u_1,\ldots,u_t$ (Each proof is correct with probability greater than $1-\frac{1}{2^K}$).

At this point G can actually find out what $u_1,\ldots,u_t$ are by activating B_i's and causing them to print u_i. This can be done since "knowing" u_i in the OT protocol actually means that u_i can be computed by running B_i twice on the same input and same coin tosses. (A simpler way to think of this is that G can read the contents of the work tapes of B_i's where u_i is written).

Recall that G is supposed to return to each B_i a random root v_i of u_i^2 mod N_A that does not separate M. G can't directly compute random square roots of u_i^2 mod N_A from u_i^2 mod N_A. But random square roots correspond, by Lemma 1, to random splits of N_A. So G can first compute (S_1,S_2) - a random split of N_A which does not separated M. He then uses the Chinese Remainder algorithm to compute v_i mod N_A such that $v_i \equiv u_i$ mod S_1 and $v_i \equiv -u_i$ mod S_2. Finally, G gives v_i to B_i.

The faulty processors $B_1,\ldots, B_t$ now have all the "relevant" information they held at the turning point of the actual protocol. They have $N_A = M \cdot p_3 \cdots p_l$, y_A and $(u_1,v_1),\ldots,(u_t,v_t)$. G can now invoke $\overline{F}$ a to predict the least significant bit of $x_A^3 = \sqrt{y_A}$ mod N_A.

If N_A is a lucky extension of M, F succeed with probability greater than $\frac{1}{2}+\frac{\varepsilon}{4}$. In other words, G succeeded in realizing an oracle for the least significant bit which is correct with probability greater than $\frac{1}{2}+\frac{\varepsilon}{4}$. By Theorem 4, G can be transformed to an efficient algorithm for inverting x^3 mod M.

As an ε fraction of all extensions of M are lucky, we will find such lucky extension in expected $O(\varepsilon^{-1})$ iterations. Thus by RIA, ε must be smaller than $1/K^c$ for every $c>0$ and sufficiently large K. **QED**

4. Implementing Simultaneous Broadcast using Verifiable Secret Sharing

We now give a high level description of a simulation of a given protocol $PROT$ which is defined (and works correctly) for an n-t-ideal simultaneous broadcast network, on a n-t-broadcast network. In other words we specify an n-t-simulator, as defined in section 2.4. Our simulator, consists of a pair of algorithms (C,D) which behave as follow.

Algorithm C takes $PROT$ as input. Throughout $PROT$, in every round j, processors $A_1,A_2,\ldots,A_n$ wish to send messages $m_{1,j},m_{2,j},\ldots,m_{n,j}$ "simultaneously".

To achieve this simultaneous delivery in a broadcast network, C replaces each original round j by a "macro" round j which involves 7 message exchange rounds in the broadcast network. In the end of this "macro"-round, processors $A_1,\ldots, A_n$ run algorithm D on the messages they received during the last 7 broadcast network rounds to compute $m_{1,j},\ldots,m_{n,j}$.

A *macro round j* consists of each processor A_i $(1 \leq i \leq n)$ running a (slightly modified, as described below) VSS protocol using its message $m_{i,j}$ ($1 \leq i \leq n$) as its secret. The (slightly modified) VSS protocol involves seven steps of message exchanges: each one of these must be done within the same time interval by each processor. A common clock is used in order to enforce a timeout mechanism. Thus, if for instance processor A_i did not broadcast a message in step 5 of the VSS on time, his message m_{ij} of the j-th macro round is is taken to be the empty messager.

The VSS protocol executed for this application must be modified as follows. Add step 5.5 to the end of part 1. In this step the dealer A, interactively proves to every other processor C that he can invert RSA with exponent 3 on the modulus N_A.

Step 5.5 is added for the following reason: all processors A_1,..., A_n are executing concurrently n VSS protocols on their respective messages (secrets) m_1,..., m_n; in step 5 of the VSS each must broadcast an N_{A_i} and y_{A_i}. These N_{A_i},y_{A_i} pairs must appear in the network channel within a certain time interval, but can arrive in random order (i.e some may appear earlier then others). Thus, a faulty processor A_f may wait till he sees the pair N_{A_p},y_{A_p} of some proper processor A_p and then choose his pair N_{A_f},y_{A_f} depending on this pair without even knowing what secret he himself is encoding. This choice of a A_f depends on the message of a proper processor A_p. This would violate the definition of an SB-network. We thus force each processor in the added step 5.5, to give a 0-knowledge (see [GMR] for definitions) proof to all other processors that he knows the content of his own message.

Th following protocol should be inserted as step 5.5 in the VSS of section 3.3.

Protocol for processor A to interactively prove to processor C that A can take cubic roots of modulo N_A

As usual let $K = |N_A|$.

1. For $1 \leq i,j, \leq K$ A picks $x_{i,j}$ at random in $Z^*_{N_A}$. A sends $x^3_{i,j}$ mod N_A to C.
2. For $1 \leq i,j \leq K$ C lets $g_{i,j}$ be his guess as to what is the least significant bit of $x_{i,j}$. C sends to A his guesses $\{g_{i,j}\}$.
3. For $1 \leq j \leq K$ A lets $y_j = g_{1,j} \cdots g_{K,j}$ mod N_A ("." denotes concatenation). For $1 \leq i,j \leq K$, A sends to C $x_{i,j}$ and z_j such that $z_j^3 = y_j$ mod N_A (z_j is the cubic root of y_j).

The above protocol constitutes a 0-Knowledge proof of A to C that A is able to take cubic roots modulo N_A.

Theorem 6 (Simultancity Theorem): Under the IRA, the compiler (C,D) described above constitute an n-$O(log\ n)$-simulator.

Rough Proof Sketch: Fix adversary ADV', protocol $PROT$ and a broadcast network NET'. This determines a probability distribution $H'_i(NET',C,D,PROT,ADV')$ on the messages delivered in the first i macro rounds. We wish to show that this distribution is indistinguishable from the probability distribution $H_i(NET,PROT,ADV)$ of messages delivered in an ISB network NET for some adversary algorithm ADV.

The outline of the proof is as follows.

For notational simplicity let the message m_{ji} being sent by the A_jth processor during the i-th macro round belong to {0,1} (essentially the same proof will work for general m_{ji}'s). Similiarly, for notational convinience, assume that exactly t processors are faulty and that processors A_1,...,A_{n-t} are proper and their messages are computed by the legal protocol $PROT$.

Let
$E_j(m_{ji}) = \{x^3 \ mod\ N_{A_j} : least\ significant\ bit\ of\ x\ is\ m_{ji}\}$
be the set of probabilistic encodings of m_{ji} using processor A_j's key N_{A_j}.

The algorithm ADV' takes as input the history $H'_{i-1} = \{m_{xj} \mid 1 \leq x \leq i-1, 1 \leq j \leq n\}$ and the probabilistic encodings $aplha_1 \in E_1(m_{1,i})$,..., $\alpha_{n-t,i} \in E_{n-t}(m_{n-t})$, and computes the i-th macro-round messages $m_{n-t+1,i}$,...,$m_{n,i}$ of the faulty processors.

Let Δ_i denote the distribution of messages output by ADV' given the same history H_{i-1} and the probabilistic encodings $\alpha_1 \in E_1(0),\ldots, \alpha_{n-t} \in E_{n-t}(0)$. Clearly, Δ_i is a distribution which can be achieved by an advresary ADV in an ISB-network.

Let $DIST$ be a polynomial-time distinguisher as defined in 2.4.1. Denote by q_1 (q_2) the probability that $DIST(m_{1,i},\ldots,m_{n,i})=1$ where the messages m_{ji} are picked according to Δ_i (and $H_i(NET,C,D,PROT,ADV)$ respectively). Assume, towards a contradiction, that $|q_1-q_2|>\frac{1}{poly(K)}$ for infinitely many security parameters K. We will show how to violate the security requirement for one of the probabilistic encryption schemes E_j (i.e violate the RSA intractability assumption). In particular, for some $1 \le j \le n-t$ we will implement a polynomial time algorithm to distinguish between the set of probabilistic encryptions $E_j(0)$ and $E_j(m_{ji})$ as follows:

> Pick $m_{1,i},\ldots,m_{n-t,i}$ of the proper processors according to protocol $PROT$. Choose $\alpha_1 \in E_1(m_{1,i}),\ldots,\alpha_j \in E_j(m_{j,i}),\alpha_{j+1} \in E_{j+1}(0),\ldots,$ $\alpha_{n-t} \in E_{n-t}(m_{n-t,i})$ at random and feed $\alpha_1, \ldots, \alpha_{n-t}$ to the adversary box ADV to compute $m_{n-t+1,i},\ldots, m_{n,i}$. Now, replace α_j by β_j picked at random in $E_j(0)$ and feed $\alpha_1,\ldots,\beta_j,\alpha_{j+1},\ldots,\alpha_n$ to the ADV to compute $m'_{n-t+1,i}, \ldots, m'_{n,i}$. Computing $D(m_{1,i},\ldots,m_{n,i})$ and $D(m_{1,i}, \ldots, m_{n-1,i}, m'_{n-t+1,i}, \ldots, m'_{n,i})$ will yield a different response and thus a polynomial time algorithm that can tell $E_j(0)$ and $E_j(m_{j,i})$ apart has been realized.

But by Theorem 4 and [GM] this contradicts the IRA. Thus, Δ_i must have been indistinguishable from $H_i(NET,C,D,PROT,ADV)$.

5. Applications to Point-to-Point Communication Networks with Byzantine Faults

We now consider the problem of simulating simultaneous broadcast on a point-to-point semi-synchronous network. For simplicity, we outline the simulation on a synchronous network.

A clarifying discussion.

Before going any further, let us stress that simulating a simultaneous network is not trivial, even if one starts from a synchronous network. The following naive attempt may be of help in clarifying the definition of simultaneity.

Given a synchronous network, let an "extended round" consist of first sending messages at prescribed times i and then using subsequent (elementary) rounds to achieve Byzantine agreement about what was sent.

> *Such extended rounds do not make the network simultaneous.*

Of course in a synchronous network all messages can be sent only at exact times 1,2,3,... and will be received exactly one unit of time later. Thus a message sent at time i is necessarily independent of the other messages sent in the same round. However, a faulty processor, P, may have sent (to different processors) two different messages m_1 and m_2, one of which is possibly the empty message. When Byzantine agreement is run, P cannot prevent the correct processors from agreeing on only one message, However, it can influence and actually decide on which of the two messages the agreement will be reached. I.e., it can decide which message to send in the extended round. Notice that, at the start of the Byzantine agreement algorithm, P already knows what messages it received from other processors. Thus what P sends in an extended round effectively depends on what other processors have send in the same round.

The Solution

The key idea is to use the power of Byzantine agreement in order to simulate a semi-synchronous broadcasting network. Once such network is constructed, apply our simulation of simultaneous broadcast network to it.

Simulating semi-synchronous broadcast by Byzantine agreement adds an $O(t)$ factor of communication rounds and a polynomial factor of computation/communication overhead.

5. Concluding Remarks

We introduced the notion of simultaneous broadcast in fault-tolerant distributed computing. We gave one implementation of this notion, under the intractability assumption of RSA.

Our implementation requires computation and communication which are exponential (2^t) in the number of faults, t. Thus to be polynomial in the size of the network, n, t can be at most $O(\log n)$ large. This leaves the question of fully polynomial implementation for larger values of t open.

Acknowledgements

We would like to thank Cynthia Dwork, Mike Fischer, Oded Goldreich, and Ron Rivest for their most helpful comments.

6. References

[ACGS]W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr, *RSA/Rabin Bits are* $\frac{1}{2} + \frac{1}{poly(\log N)}$ Secure, Proc. of 25th FOCS, 1985.

[ABCGM]B. Awerbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali, *How to Implement Bracha's $O(\log n)$ Byzantine Agreement Algorithm*, unpublished manuscript, 1984.

[Bl1] M. Blum, *Coin flipping by telephone*, IEEE COMPCON 1982.

[Bl2] M. Blum, *Three applications of the oblivious transfer*, Unpublished manuscript, 1981

[Br] G. Bracha, *An $O(\log n)$ Rounds Randomized Byzantine Agreement Algorithm*, Proc. of 17th STOC, 1985.

[BPT]R. Berger, R. Peralta, and T. Tedrick, *On fixing the Oblivious Transfer*, Presented in Eurocrypt 1984.

[C] B. Chor, *Two Issues in Public-Key Cryptography*, Ph.D. dissertation, MIT, 1985.

[CGG]B. Chor, S. Goldwasser, and O. Goldreich, *The Bit Security of Modular Squaring Given Partial Factorization of the Modulus*, to appear in Proc. of Crypto 85, 198?.

[EGL]Even, Goldreich Lempel, *A randomized protocol for Signing Contracts*, CACM, Vol. 28 No. 6, 1985.

[FMR]M. Fischer, S. Micali and C. Rackoff, *A Secure Protocol for the Oblivious Transfer*, Eurocrypt 1984.

[GM]S. Goldwasser, and S. Micali, *Probabilistic Encryption*, JCSS Vol. 28 No. 2, 1984.

[GMR]S. Goldwasser, S. Micali, and C. Rackoff *The Knowledge Complexity of Interactive Proof Systems*, Proc. of 17th STOC, 1985.

[HR] J. Halpern and M.O. Rabin, *A Logic to reason about likehood*, Proc. of 15th STOC, 1983.

[R1] M. Rabin, *Digitalized Signatures and Public Key Functions as Intractable as Factoring*, Tech. Rep. MIT/LCS/TR-212, MIT, 1979.

[R2] M. Rabin, *How to Exchange Secrets by Oblivious Transfer*, unpublished manuscript, 1981.

[RSA]R. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, CACM Vol. 21 No. 2, 1978.

[S] A. Shamir, *How to Share A Secret*, CACM Vol. 22 No. 11, 1979.

[Y1] A. Yao, *Theory and Applications of Trapdoor Functions*, Proc. of 23rd FOCS, 1982.

[Y2] A. Yao, private communication.