MULTIPARTY UNCONDITIONALLY SECURE PROTOCOLS

(Extended Abstract)

*David Chaum** *Claude Crépeau*† *Ivan Damgård*‡

*Centre for Mathematics and Computer Science (C.W.I.)
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

†Laboratory for Computer Science, M.I.T.
545 Technology Square, Cambridge, MA 02139, U.S.A.

‡Matematisk Institut, Aarhus Universitet,
Ny Munkegade, DK 8000 Aarhus C, Denmark

Abstract

Under the assumption that each pair of participants can communicate secretly, we show that any reasonable multiparty protocol can be achieved if at least $\frac{2n}{3}$ of the participants are honest. The secrecy achieved is unconditional. It does not rely on any assumption about computational intractability.

1. Introduction

The problem of multiparty function computation is as follows: n participants $P_1, P_2, ..., P_n$ agree on a multivariable function F and wish to compute and reveal to each participant $z=F(x_1, x_2, ..., x_n)$, where x_i is a secret input provided by P_i. The goal is to preserve the maximum privacy of the x_i's and to simultaneously guarantee the correctness of the common result z. (An intrinsic property of any solution to this problem is that for a non-trivial function F, the value of z reveals some information about the secret x_i's.)

In this paper, we show that essentially any general multiparty protocol problem can be solved, in such a way that each party's secrets is unconditionally secure, assuming the existence of authenticated secrecy channels between each pair of participants. In general, an input value x_i is unconditionally secure if gaining information about x_i is impossible beyond that available from z (so long as no more than $\frac{1}{3}$ of the participants cheat, in our model). This is stronger than the notion of cryptographic security that is often used for cryptographic protocols. Under that definition, x_i is cryptographically secure if gaining information about it, other than that available from z is thought to be computationally hard.

As explained below, in our model no more than $\frac{1}{3}$ of the participants may deviate from the protocol. Since our solution will tolerate up to this number of participants who cheat, it is therefore optimal.

1.1. Related Work

Other work has been able to provide unconditional privacy in multiparty protocols, but only for specific problems. One poker protocol of Bárány and Furedi [BF] used a model similar to ours, but was unable to tolerate active cheaters. The dining cryptographers problem of Chaum [Ch2], was also based on a similar model, and provided unconditional untraceability of messages even in the presence of active disruption.

The general problem of achieving secure multiparty function computation was first posed by Yao

† supported in part by an NSERC Postgraduate Scholarship
supported in part by DARPA grant N0001483K0125
research conducted at the C.W.I.
‡ research conducted at the C.W.I.

[Ya], in a public key cryptographic setting. In this paper, he suggested that a general solution exists in his particular model. Goldreich, Micali and Wigderson showed in [GMW] that very general multiparty protocols (or mental games) could be achieved in a model where security is based on the notion of zero-knowledge protocols. Their solution, based on the existence of trapdoor one-way permutations, involves a "compiler" that transforms any mental game into a multiparty cryptographically secure protocol.

Chaum, Damgård and Van de Graaf presented a more direct and practical solution [CDG] based on the existence of "unconditionally secure blobs" (see [BCC]) and trapdoor one-way functions. This solution was the first one to raise the hope that such protocols could be implemented in an unconditionally secure way. That paper showed how general multiparty protocols provide unconditional privacy to one participant and that it is the most that can be achieved in that model. Our result stems from that paper.

The major limitation of these results is due to the model they use: a setting where only public communications are possible. All these general constructions rely on trapdoor one-way functions, and therefore must assume essentially that public key cryptography is possible.

A much weaker assumption is to assert the existence of authenticated secrecy channels, i.e. a way of communicating in which the identity of the sender is known (authentication) and the data transferred is revealed only to the single person it is meant for (secrecy). Such channels are very practical and can be implemented easily: for example, by writing down messages on pieces of paper and physically handing them out to the other parties. They can also be implemented using conventional cryptography (secret key systems).

Our work has drawn inspiration from and relies on a number of other earlier contributions. The Byzantine Generals problem proposed and solved by Lamport, Shostak and Pease [LPS] can be thought of as underlying our work and provided a foundation for our model. So called secret sharing schemes proposed by Blakely [Bl] and Shamir [Sh] are basic building blocks. A very clever extension of these schemes was proposed by McEliece and Sarvawate [MS] that provides some fault-tolerance to active cheaters. The more specific notion of verifiable secret sharing (VSS) schemes was introduced to cryptography by Chor, Goldwasser, Micali and Awerbuch [CGMA]. The usefulness of the homomorphic structure of Shamir's secret sharing scheme was observed by Benaloh [Be], who proposed techniques very similar to ours in is own version of a VSS scheme.

Some concurrent and independent work [BGW] has been performed on the topic of our paper: during discussions with Shafi Goldwasser and Avi Wigderson, we learned that they were working with Michael Ben-Or on results similar to ours. At that time, all of us had results in a very early stage. By the time of submission to this conference, both groups had found almost identical results by quite different means.

1.2. Algorithm

The general structure of the algorithms is similar to the ones of [CDG] & [GMW] in the sense that it takes place in two steps: Commitment and Computation. First the participants enter a stage in which they commit to their inputs. This commitment is performed by means of a new non-cryptographic verifiable secret sharing (VSS) scheme. Up to now, all previous implementation of VSS schemes have relied on public key cryptography. We introduce the first scheme that does not rely on such assumptions.

If some participants try to commit to something improper or simply do not cooperate, this first phase will identify them and the remaining participants will take the appropriate action. This is the very best we can hope for. What else can you do with someone who does not want to participate? Once every one commits to his input, and that every participant gets a share of everybody else's secret, they enter the second phase in which they evaluate the function. The computation is performed locally by each participant on the shares he receives from the others.

Our construction satisfies the following properties:

- Unconditional Secrecy: In both stages, it is impossible for any subset of participants of size less than $\frac{n}{3}$ to gain any information about anyone else's inputs.
- Built-In Fault Tolerance: In the second phase, no such subset can prevent the honest participants from correctly evaluating the function.

Again, our solution does not depend on restricting the computing power of the participants. Earlier solutions relied on cryptographic assumptions for both secrecy of the inputs and correctness of the computation. Even if these assumptions turned out to be true, the secrecy and correctness would still be dependent on the limitations in computing power of the participants.

2. The Model

For convenience, the number of participants will be called n, which can always be written as $n = 3d + a$, where $a = 1, 2$ or 3.

Our assumptions about at least $2d+a$ of the participants are that:

- they do not leak secret information to other participants; and
- they send the correct messages defined by the protocol.

We call a participant satisfying the above properties *reliable*. At the start of the protocol, it is of course not generally agreed which participants are reliable. Let $P_1, P_2, \cdots, P_n$ be the participants. Our basic assumptions about the communication between reliable participants P_A and P_B are that:

- when P_A sends a message to P_B, nobody else can learn anything about its content;
- when P_B receives a message from P_A, P_B can be certain that nobody but P_A could have sent the message; and
- messages sent will be received in a timely manner.

Finally, we complete our model by assuming the following:

- all participants agree on the protocols to be followed; and
- participants can determine whether messages sent to them were sent before deadlines set in the protocol.

Our protocols ensure that all reliable participants obtain the correct result. It is proved constructively in [LPS], under a model like ours, that a necessary and sufficient condition for all reliable participants to agree on a message—such as the result of a protocol—is that at least $2d+a$ of the participants are reliable. Hence, our two-thirds assumption is optimal. A polynomial algorithm solving this problem is presented in [DS]. Their construction allows us to obtain an efficient "broadcast" channel: a means allowing any participant to make a message known to all participants, in such a way that all reliable participants will obtain the same value of the message. (Assuming a broadcast channel, moreover, would not enable us to to weaken our other requirements.)

For simplicity in the following descriptions, we use the terminology of information theory because we make the assumption that the channels are unconditionally secure. Notice however that in fact we get protocols as strong as the secrecy and authentication of the channels used. If the channels were not unconditionally secure, for example, the protocol would not be unconditionally secure for all participants but its correctness would still be guaranteed.

3. Implementing Blobs using Secret Sharing

In [BCC], a fundamental protocol primitive is described: *the blob*. The purpose of blobs is to allow a participant P_A to *commit* to a bit in such a way that she cannot later change her mind about the bit, but nobody else can discover it without her help. The defining properties of blobs are as follows:

(i) P_A can obtain blobs representing 1 and blobs representing 0.

(ii) When presented with a blob, nobody can tell which bit it represents.

(iii) P_A can *open* blobs by showing the other participants the single bit each represents; there is no blob she is able to "open" both as 0 and as 1.

(iv) Any other participant can at will obtain blobs representing 0 and 1. Moreover, these blobs must look exactly like the blobs obtained by P_A.

To implement blobs in our model, we use a variation on Shamir's secret sharing scheme [Sh]. This variation was proposed by Blakely [Bl], who independently discovered secret sharing schemes, and it is more efficient than Shamir's original construction.

For our purposes, the scheme may be described as follows: a polynomial f of degree at

most d over $GF(2^k)$ is chosen uniformly, where k is an integer such that $2^k > n$. The secret to be shared is defined for convenience as the value of f at 0. The protocol also assigns a distinct non-zero point i_B in the field to each participant P_B. The secret can now be divided among the n participants by providing each P_B with the value of $f(i_B)$. It is not hard to see that more than d shares completely determine f, and therefore the secret, while no Shannon information about the secret is revealed by any number of shares not exceeding d.

We generalize slightly by allowing blobs to represent any value in $GF(2^k)$. Blobs are now readily achieved:

(i) To obtain a blob representing the value v, participant P_A chooses uniformly a polynomial f with $deg(f) \leq d$, such that $f(0) = v$. She then calculates n shares as above and distributes one to each participant. Using the subprotocol described below, she convinces the other participants that she has distributed a consistent set of shares.

(ii) Since the number of unreliable participants is smaller than d, no collusion will gain any information in the Shannon sense about the value represented by a blob.

(iii) To open a blob, P_A first broadcasts what its shares should be $(\{i_B | 1 \leq B \leq n\})$. Then each participant broadcasts a message stating whether they agree with their share that was broadcast by P_A. If a participant does not agree, he is said to be *complaining* about P_A. It is required that at least $2d+a$ of the participants do not complain. By the remarks below, this condition ensures that P_A can only open a blob to reveal the single value it represents.

(iv) Any participant can choose a polynomial and distribute shares of it, whence it is impossible to tell from a blob who generated it.

By distributing inconsistent shares to reliable participants, a coalition of unreliable participants could allow P_A to open a blob in two or more different ways. The following proof, which we informally call a "cut-and-choose procedure" (and is similar to the construction of [Be]) enables us to remove this inconsistency. Let the original blob chosen by P_A be β. Then the cut-and-choose works as follows:

(a) P_A establishes a new independently chosen blob δ.

(b) One of the other participants flips a coin and asks P_A to
- open δ, or to
- open $\delta+\beta$, where $\delta+\beta$ denotes the blob defined by the sum of corresponding shares of δ and β.

(c) Steps (a) and (b) are repeated until no complaints have occurred in m consecutive rounds, or until more than d participants have complained about P_A. In the first case the proof is accepted, otherwise it is rejected.

The participants take turns in executing step (b). By assumption, this means that P_A will be unable to predict the coinflips at least $\frac{2d+a}{n}$ of the time.

Note that the proof will always terminate: even if all unreliable participants work against an honest P_A, they cannot enlarge the number of rounds by more than md.

When β is later opened, the shares held by complaining participants are of course ignored.

If the proof is accepted, then the following holds with probability exponentially close to 1 in m: all reliable participants who did not complain (of which there are at least $d+a$) have shares consistent with one polynomial of degree at most d.

Thus, with very high probability, P_A cannot convincingly claim that her blob contains anything but the secret determined by the $d+a$ valid shares guarantied by the fact above, since otherwise the condition in step (iii) would be violated.

To see why this is satisfied, it suffices to consider the behavior of reliable participants, corresponding to the worst case assumption that all unreliable participants will try to help P_A by always agreeing with her. For any blob γ, consider a polynomial consistent with a maximal number of shares of γ, and let $C(\gamma)$ be the number of remaining shares held by reliable and non complaining participants. Thus $C(\gamma)$ may vary over time. In other words, no matter how P_A tries to open γ, at least $C(\gamma)$ participants will complain. The case where P_A created γ correctly corresponds of course to $C(\gamma) = 0$.

In any of the rounds of the subprotocol above, it is easy to see that because the sum of δ and $\delta+\beta$ is

just β, $C(\delta)+C(\delta+\beta) \geq C(\beta)$ must hold. So if at any point $C(\beta) > 0$, then P_A cannot go through m rounds without complaints unless she can predict roughly $\frac{2m}{3}$ coinflips.

In [BCC], it is shown how one can construct, using only blobs, efficient *minimum disclosure proofs* for membership in a very large class of languages, including NP and BPP. Since we can construct blobs in our model, we can also perform all such proofs directly.

4. VSS and Fault Tolerant Blobs

When opening a blob, P_A was to broadcast the shares she distributed in creating it. If P_A is trying to prove some statement using the techniques of [BCC], the previous section's results imply that it is in P_A's interest to create and broadcast the shares properly. But in other cases, communication failures or a change of heart, for example, might keep P_A from ultimately broadcasting the shares. Even if the other participants were to make P_A's shares public in efforts to open the blob without P_A's help, they would be left with a computational problem: unreliable participants might make public false values for their shares, and finding the value represented may require searching the exponentially many subsets of shares of size $2d+a$ for one consistent with one polynomial of degree smaller than d. Even worse, if P_A was already cheating when she created the blob, the majority of complainers could be reliable. In such cases, unreliable participants could choose at the time of opening between broadcasts that would leave no unique solution for the secret or other broadcast that would yield a particular value unambiguously.

This is where the secret sharing scheme becomes insufficient and a VSS is needed. To avoid the problems mentioned above, and assist with things to be presented later, we provide for the "sharing of the shares of a blob" (as was done for similar reasons in [Ch]). Thus, to create a *double* blob δ, P_A proceeds as follows:

(1) She creates an ordinary blob in the same way as in the previous section. This blob is called the *top level* blob, and contains the secret she commits to.

(2) For each participant P_B. the following is done: suppose P_A sent the share s_B of her original blob to P_B. Then P_B creates a *sub-blob*, i.e. he creates a blob β_B containing his share s_B.

(3) By the remarks in the previous section, all participants are now committed to their share of the top-level blob. A cut-and-choose procedure is now used to check that everybody has committed to the proper share: P_A creates a number of additional double blobs $\delta_1, \delta_2, \cdots, \delta_t$ (for which each participant creates his own sub-blobs), and according to coin flips made by other participants, either all shares of the new double blob are made public or the sum of corresponding shares of the new and the original double blob are broadcast. Thus in each round, every participant opens a sub-blob of his own (either a new one or a sum) to confirm his agreement or disagreement with P_A on what she sent him originally. In order for the proof to be accepted, a subset consisting of at least $2d+a$ participants must agree with P_A in all rounds. If a participant disagrees with P_A at any point, then his share and sub-blob will be ignored when the original double blob is later opened.

It is easily seen that if the proof in (3) above is accepted, then the following holds with probability exponentially close to 1 in the number of coin flips:

- all sub-blobs accepted by the cut-and-choose contain a uniquely defined share of the top-level blob; and
- all these shares are consistent with one polynomial.

To open a double blob, all participants broadcast their shares of the top level blob as well as all shares of their sub-blobs. The result of the opening is uniquely and easily determined, since in this case the effect of the sub-blobs is to prevent unreliable participants from issuing improper shares of the top level blob: if a participant cannot confirm his share by opening his sub-blob correctly, it will just be ignored.

5. Multiparty Computations

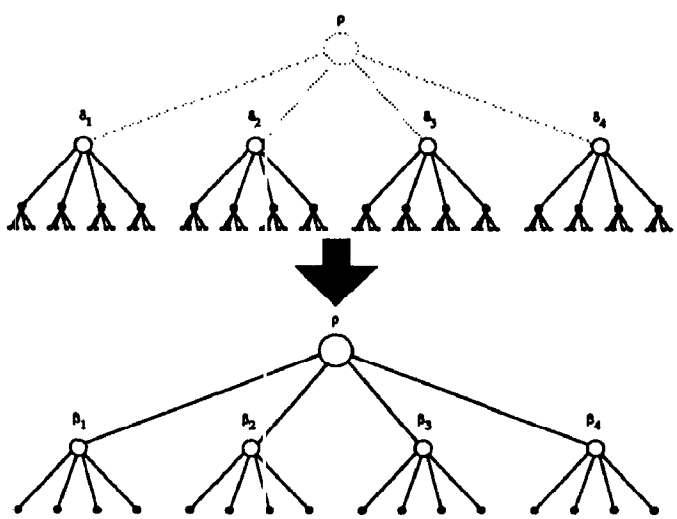
This section considers general multiparty computations. These may involve secret input from each participant, and a single output which should become known to all reliable participants.

In the first step of the protocol, all participants commit to their secret input bits by distributing shares of them to all participants. The basic idea is now to do the computation by having each participant perform a corresponding computation on the shares he received. There are two problems with this idea: first, we cannot trust all participants to do the correct computation. Therefore participants must be committed to their shares, so that they can prove that the protocol was followed. This suggests a structure similar to that of a double blob. Secondly, for technical reasons explained later, all reliable participants must be able to complete the computation on their shares. Thus we cannot tolerate any complaints about the shares distributed, since there may be no way to tell whether a complainer is reliable or not. This leads to the following definition of a *robust* double blob:

- like a double blob, a robust double blob has a top-level blob and sub-blobs, where the top-level contains the bit committed to.
- *all* sub-blobs contain valid shares of the top-level blob.

The double blob as described in the previous section clearly does not always satisfy these properties. We can, however, get robustness by using the fact that a double blob, once verified by cut-and-choose, can always be opened without the help of its creator, and even in spite of unreliable participants. First, notice that the content of a top-level blob is completely determined by the shares of the sub-blobs (called sub-shares), if these are consistent. Thus, to create a robust double blob ρ, P_A creates a set $S=\{\delta_1,\delta_2,\cdots,\delta_n\}$ of double blobs, each one is supposed to contain a share of ρ (note that the sub-blobs in δ_B are created by P_B after receiving shares from P_A) Once each δ_B is verified as in the previous section, it is opened to P_B. Remember that this operation can be achieved without the help of P_A. At this point P_B commits to the share hidden in the double blob δ_B using a single blob β_B. A gigantic cut-and-choose is then used over this structure to prove its correctness. Two things have to be proven about this structure:

- All double blobs in S contain shares of ρ consistent with one polynomial.



Structure of a robust double blob

- Each P_B has committed to the same share as is contained in the double blob P_A made for him ($Contents(\beta_B) = Contents(\delta_B)$).

We leave it as an exercise to design a cut-and-choose procedure that will establish this fact.

Note that this protocol leaves no possibility for P_A to cheat and blame the resulting disagreement on some other participant: if less than d participants complain about P_A, then a valid commitment has (with very high probability) been constructed, and otherwise it is obvious that P_A is unreliable.

When this first phase including creation of commitments for all input bits and proofs of validity is completed successfully, the protocol is fault-tolerant: there is no way the unreliable participants can stop any reliable participant from computing the correct result.

The computation is specified by a boolean circuit composed of XOR and AND gates. It is then clearly sufficient to be able to safely compute from two robust double blobs a new one both as the XOR of the two inputs and also as the AND.

Computing the XOR of two double blobs is easy, based on the remarks in previous sections: all participants simply add their shares, both for the top-level blobs representing the actual bits, and for the sub-blobs. The outcome is just a new double blob representing the XOR of the inputs.

Basically, computing the AND is just as simple: the participants merely multiply the shares. But this raises some technical problems, since the computation involves polynomials of degree larger than d; polynomials of this large degree will not be robust enough against unreliable participants.

Consequently, the AND is instead done in two steps:

(1) Each participant multiplies his shares of the two top-level blobs and commits to the product using a sub-blob. He then proves by a cut-and-choose (to be described below) that the multiplication was done correctly.

The result of (1) is a double blob containing the AND of the two bits, but with a large degree polynomial in the top-level blob. We cannot continue the computation with this blob, since for one thing the degree would eventually grow too large for the secrets to be uniquely determined. Therefore, this degree is brought down below d as follows:

(2) Each participant chooses a pair of robust double blobs constructed as in the beginning of this section, and such that the top level involves a pair of randomly chosen polynomials (f,g), where $deg(f) < 2d$, $deg(g) < d$, and $f(0) = g(0) \in \{0,1\}$. We leave as an (easy) exercise construction of a cut-and-choose for proving correctness of such a pair. When all these pairs are added, the result will be a pair still satisfying the conditions above, but such that nobody knows the common value of f and g in 0. Finally, the double blob constructed with f is x-ored with the one computed in (1), and the result is opened. If this result is 0, then the computation continues with the blob from g, otherwise $1+g$ (the complemented bit) is used.

We have now only to describe the cut-and-choose mentioned in (1). In principle, this procedure is essentially the same as the computation protocols of [BC]: the prover has committed to s_1, s_2 and s_3, and claims that $s_1 s_2 = s_3$. He then commits to a row-permuted version of the multiplication table for the field used. The other participants, responsive to their coin flips, now ask him either to open the entire table or to prove that one of the rows contains commitments to the tuple (s_1, s_2, s_3). This is repeated to attain the desired level of certainty. Note that since the size of the field need only just exceed n, only a number of messages quadratic in n are sent.

We call attention to the possibility of a trade off between vulnerability to disruption and efficiency of the protocol. The initial commitment phase could in fact be completed correctly using only ordinary double blobs, if we require that *nobody* complains about anybody during the initial phase. This requirement is easily seen to imply that all the double blobs constructed are (with very high probability) robust. With this method, however, it is not possible to find out who has not been following the protocol in the first phase, if complaints do occur.

6. Generalizations

The one third assumption on the number of unreliable participants is necessary to ensure that Byzantine agreement is possible. It is natural, however, to ask what can be done if we ensure this simply by assuming the existence of a broadcast channel as part of the model?

In fact, even with this assumption, it is impossible to implement unconditionally secure blobs while tolerating more than d unreliable participants. Informally, this is so, since if P_A tries to commit to some secret, she must send a set of messages containing enough Shannon information to determine her secret completely. She cannot use the broadcast channel for this, since her secret would then become public immediately. Moreover, if there are U unreliable participants, then no subset of this size or smaller must be given enough information to determine the secret, since the set of unreliable participants is unknown. When later the participants try to determine which secret P_A is in fact committed to, the unreliable participants are free to fabricate some set of messages which they will claim P_A sent them originally. Since any subset of U messages leaves the secret completely undetermined, it is easy to construct the set of false messages such that it is consistent with the messages sent to U reliable participants. We thus have a situation, where $2U$ participants seem to agree on something, while the rest, say R, participants are complaining. But if we allow $U > n/3$, then $R \leq U$, and thus there is no way of finding out whether the situation is in fact as described above, or the R participants are just unreliable ones, complaining for no good reason! Moreover, this ambiguous situation could result, even if P_A has followed the protocol.

It is also possible to tolerate more unreliable participants, if we change the model by restricting their behavior. If we assume that no participant will ever send an incorrect message during the protocol, then two forms of behavior remain, that may cause problems in the protocol:

1) Sharing secret information with other participants; and

2) Stopping the protocol too early.

In the following, assume that at least C participants will complete the protocol, while at most L participants will leak secrets to others.

Clearly, information about the inputs to a computation must be distributed in such a way that any subset of C participants or more can recover all inputs, since otherwise there is no guaranty that the computation can be completed. But if the inputs are to remain unconditionally protected, this means that we must have $L < C$.

One can now make the simplifying assumption that the set of participants is partitioned in one subset in which participants may show both forms of unreliable behavior mentioned above, and another subset, where there is no deviation from the protocol at all. This means that $C+L = n$, and therefore that $L = \lfloor (n-1)/2 \rfloor < C$. Hence the best a protocol can hope to do in this case is to tolerate the situation where $L = \lfloor (n-1)/2 \rfloor$ and $C = n-L$. But this can easily be accomplished using our basic protocol with polynomials of degree L. Because of the inequality on L, multiplication of polynomials will not lead to loss of information. As usual, protection against early stopping is effective after the initial commitment phase, where double blobs are used. If a participant stops, the remaining ones can use the corresponding subshares as input to a separate instance of the basic protocol which will simulate the missing participant.

Without the assumption that $C+L = n$, things seem to become more complicated. It is clear that as long as $L \leq \lfloor (n-1)/2 \rfloor$, then the solution outlined above still works, but without this condition, it is not clear what happens. The method with multiplication of polynomials does not work any more, because it leads to polynomials of a degree larger than the number of available shares. Therefore the construction of a general computation protocol under these special assumptions remains an open problem.

7. Open Problem

Is it possible to extend our result such that more unreliable participants can be tolerated if we are willing to revert to a cryptographic assumption in the case where $n/3 < U < n/2$. (Therefore achieving an "Obliviously cryptographic" multiparty unconditionally secure protocol.)

8. Acknowledgements

We would like to thank Robert Ashcroft, Gilles Brassard, Ernie Brickell, Shafi Goldwasser, Jeroen van de Graaf, Silvio Micali, Micheal Sacks and Avi Wigderson for the discussions and suggestions about this paper and their interrest in our result.

References

[BF] Bárány and Furedi: Mental Poker with Three or More Players, Information and Control, vol. 59, 1983, pp.84-93.

[Be] Benaloh: Secret sharing homomorphisms, Proc. of Crypto 86.

[Bl] Blakely: Security proofs for information protection systems. Proceedings of the 1980 Symposium on Security and Privacy, IEEE Computer Society Press, NY, 1981, pp.79-88.

[BC] Brassard and Crépeau: Zero-Knowledge Simulation of Boolean Circuits. Proceedings of Crypto 86.

[BCC] Brassard, Chaum and Crépeau: Minimum Disclosure Proofs of knowledge. To appear.

[BGW] Ben-Or, Goldwasser and Wigderson: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. to appear in Proceedings of STOC 88.

[Ch] Chaum: How to keep a secret alive. Proceedings of Crypto 84.

[Ch2] Chaum: The Dining Cryptographers Problem, to appear.

[CDG] Chaum, Damgård and van de Graaf: Multiparty Computations ensuring secrecy of each party's input and correctness of the result. To appear in Proceedings of Crypto 87.

[CGMA] Chor, Goldwasser, Micali and Awerbuch: Verifiable Secret Sharing and Achieving Simultaneity in the Presence of faults. Proceedings of FOCS 85, pp.383-395.

[DS] Dole and Strong: Polynomial Algorithms for Multiple Processor Agreement. Proceedings of STOC 82, pp.401-407.

[GMW] Goldreich, Micali and Wigderson: How to play any mental game. Proceedings of STOC 87, pp.218-229.

[LPS] Lamport, Shostak and Pease: The Byzantine Generals Problem. ACM trans. Prog. Languages and Systems, vol.4, no.3, 1982, pp.382-401.

[MS] McEliece and Sarvawate: On Sharing Secrets and Reed-Solomon Codes, CACM, Vol. 24, no. 9, 1981, pp. 583-584.

[Sh] Shamir: How to share a secret. CACM, vol.22, no.11, 1979, pp.612-613.

[Ya] Yao: Protocols for secure computations, Proc. of FOCS 82, pp.160-164.