Secure Computation with Partial Message Loss

Chiu-Yuen Koo

Dept. of Computer Science, University of Maryland, College Park, USA
cykoo@cs.umd.edu
Supported by NSF Trusted Computing grant #0310751

Abstract. Existing communication models for multiparty computation (MPC) either assume that *all* messages are delivered eventually or *any* message can be lost. Under the former assumption, MPC protocols guaranteeing output delivery are known. However, this assumption may not hold in some network settings like the Internet where messages can be lost due to denial of service attack or heavy network congestion. On the other hand, the latter assumption may be too conservative. Known MPC protocols developed under this assumption have an undesirable feature: output delivery is not guaranteed even only one party suffers message loss.

In this work, we propose a communication model which makes an intermediate assumption on message delivery. In our model, there is a common global clock and three types of parties: (i) Corrupted parties (ii) Honest parties with connection problems (where message delivery is never guaranteed) (iii) Honest parties that can normally communicate but may lose a small fraction of messages at each round due to transient network problems. We define secure MPC under this model. Output delivery is guaranteed to type (ii) parties that do not abort and type (iii) parties.

Let n be the total number of parties, e_f and e_c be upper bounds on the number of corrupted parties and type (ii) parties respectively. We construct a secure MPC protocol for $n > 4e_f + 3e_c$. Protocols for broadcast and verifiable secret sharing are constructed along the way.

1 Introduction

The study of secure multiparty computation (MPC) was initiated by Yao[26] in the 2-party setting and extended to the multiparty setting by Goldreich, Micali and Wigderson[16]. Roughly speaking, a set of n parties wants to jointly compute a function g of their (private) inputs. However, up to t of them are corrupted by an adversary. The requirements are that (i) non-corrupted parties obtain their outputs and (ii) the adversary learns nothing but the outputs of the corrupted parties.

Several communication models are considered in the current body of work, giving rise to different feasibility results, as follows.

1. *Common global clock and message delivery within bounded time:* This is the synchronous model. Protocol execution consists of rounds. It is assumed that

the duration of one round is sufficient for a message to be sent and delivered from one party to another.

If $t < n/3$, then information-theoretically secure MPC protocols exist[3, 11] in a point-to-point network. If we assume the existence of a broadcast channel, then information-theoretically secure MPC protocols exist for $t < n/2$[22, 12, 1].

If the definition of secure MPC is relaxed such that non-corrupted parties are not guaranteed to receive their outputs (i.e., without guarantee on output delivery), then computationally secure MPC protocols exist[16, 17] for any $t < n$.

2. *Eventual message delivery without bound on delivery time:* This is the asynchronous model with eventual message delivery assumption. There is no assumed bound on the network latency. Under this communication model, information-theoretically secure MPC protocols exist for $t < n/3$[2, 4, 7].[1]
3. *No eventual message delivery:* This is known as the "message blocking" model and is the communication model considered in [8, 10]. There is no assumed bound on the network latency and any message can be lost. Assuming a common reference string, an Universally Composable[8] secure multiparty computation protocol (without guarantee on output delivery) exists for any $t < n$ in the computational setting[10].
4. *Local clock with bounded drift and message delivery within bounded time:* This is the timing model considered in [18]. It is assumed that the local clocks of the parties proceed at the same rate and an upper bound is known on the network latency. Under this model, an universally composable secure multiparty computation protocol (without guarantee on output delivery) exists for any $t < n$ in the computational setting[18]. It is worth mentioning that the security of the protocol holds as long as the assumption about local clocks holds. The network latency assumption is used to ensure non-triviality of the protocol.

We note that the results mentioned in models 1-3 hold for an adaptive adversary while the result mentioned in model 4 holds for a static adversary.

1.1 Applicability of existing models to general network setting

We discuss whether the assumptions made in the existing models are applicable to general network settings like the internet.

- Message delivery within bounded time: In a real network setting, an upper bound on the network latency can be very large. Even worse, as noted in [18], any reasonable bound is unlikely to hold, and hence the security of a protocol can be compromised. Consider the following scenario: n parties execute the protocol by Ben-Or, Goldwasser and Wigderson[3] and the adversary

[1] We note that this result cannot be translated to the synchronous model since the definition of secure asynchronous computation is different from the synchronous counterpart.

corrupts $n/6+1$ parties. In the first round, parties share their private inputs using a $(n/3+1)$-out-n secret sharing scheme. Suppose an uncorrupted party p_i suffers network congestion: $n/6$ uncorrupted parties fail to receive their shares of p_i's input in time. These $n/6$ uncorrupted parties will broadcast complaints in the next round and p_i will reveal the corresponding shares in the round after the next round.[2] The adversary will then have enough shares to reconstruct the private input of p_i.
- Eventual message delivery: This is a weaker assumption than the previous one. Under this assumption, we can have secure MPC protocols that guarantee output delivery. However, given the current form of the internet, this assumption may still be too strong. Messages sent to a party can be lost due to denial of service(DoS) attack[19] or heavy network congestion.
- Messages can be blocked: Under the message blocking model, known MPC protocols have an undesirable feature: output delivery cannot be guaranteed when one party suffers message loss (even if all parties are honest). An adversary can then have a simple strategy to prevent parties from receiving the outputs: carrying out a denial of service attack on a chosen party.
  The assumption that *every* message can be lost may be conservative. Depending on the scenarios, it may be reasonable to assume a few, but not many, parties suffer from from DoS attack or network congestion at the same time.

In this work, we propose a communication model that is an intermediate between eventual message delivery model and message blocking model. Under this model, we construct a secure multiparty computation protocol that guarantees output delivery to all parties except those experience severe message loss.

1.2 Our model

Three assumptions are made in our model:

1. *A common global clock:* Given the current state of art for modern network, we believe it is reasonable to assume the existence of a common global clock[3]. Protocol execution consists of rounds. Every party knows when a round starts and ends.
2. *Three type of parties:* We assume that there are three types of parties in the network:
   - Corrupted parties who are controlled by an adversary.
   - Honest parties with connection problems (where message delivery is never guaranteed). An honest party that fails to contact the common global clock belongs to this category.
   - Honest parties that can normally communicate but at each round they may fail to send/receive a small fraction of messages due to transient network problems.

[2] We remark that this is a simplification of what actually happens.

[3] For instance, NIST has provided such a service: http://tf.nist.gov/timefreq/service/its.htm?

From now on, we will address the second type of parties as constrained parties and the third type of parties as fault-free parties. A constrained party does not necessarily realize that it suffers from connection problem.

3. *A time bound related to network latency:* We assume that there is a time bound Δ such that
   - Duration of a round is equal to Δ.
   - Any fault-free party p can successfully communicate with all but δ fraction of fault-free parties in any round (i.e., message transmission from it to another party takes time less than Δ and vice versa). The set of fault-free parties p can communicate with may vary each round.

1.3 Discussions of our model and related works

The first assumption is also made in the synchronous model.

The second assumption is inspired by the previous work in distributed computing. Thambidurai and Park[24], and independently Garay and Perry[14], introduced the concept of hybrid failure model which allows a mix of different degrees of failures. Our second assumption can be viewed as assuming a mix of omission[21, 20] and Byzantine failures, which is a more general assumption than the previous ones considered in the literature.

In [23, 25, 6], protocols for broadcast and consensus are considered in a communication model where the edges can be faulty (in addition to faulty nodes). In [9], Canetti, Halevi and Herzberg considered the problem of maintaining authenticated communication over untrusted communication channels, where an adversary can corrupt links for transient periods of time. In both lines of work, there is a bound on the number of faulty links connected to an honest party. On the other hand, our model captures the scenario when an honest party suffers from *arbitrary* message loss.

We believe the third assumption is more realizable than assuming a time bound on the maximum latency, and yet it is sufficient to guarantee output delivery of fault-free parties.

If constrained parties are absent, then our model is reduced to the standard synchronous model. On the other hand, if fault-free parties are absent, then our model can be viewed as a message blocking model with time-out.

1.4 Our results

We define secure multiparty computation under our communication model. We defer the formal definition to the next section, but roughly speaking, we require the followings: (i) the fault-free parties always receive their outputs; (ii) if a constrained party does not realize that it suffers from connection problems, then it will receive its output, otherwise, it aborts; (iii) the adversary learns nothing but the outputs of the corrupted parties.

We consider an adaptive, rushing adversary. The adversary is adaptive in the sense that in any round, it can turn a fault-free party into a constrained party or into a corrupted party. In each round, the adversary has the power to decide the set of messages a constrained party can receive/send and the set of parties a fault-free party can communicate with (subject to the δ constraint). We assume each pair of parties is connected by a secure channel.

Let e_f be an upper bound on the number of corrupted parties; e_c be an upper bound on the number of constrained parties; n be the total number of parties. For $\delta < \frac{1}{6}$, we construct an information-theoretically secure MPC protocol for $n > 4e_f + 3e_c$. We define broadcast and verifiable secret sharing (VSS) under the new communication model along the way. The results are as follows (all results hold for $\delta < \frac{1}{6}$):

– A broadcast protocol for $n > 3e_f + 2e_c$; we also have a different broadcast protocol for $n \geq 3e_f + 2e_c$ if it is known that $e_f, e_c \geq 1$.
– A VSS protocol for $n > 4e_f + 3e_c$.

2 Notations, Definitions and Overview

2.1 Notations

We use two special symbols ϕ and $\perp$ in the paper. ϕ is a special symbol denoting the failure of receiving a valid message. During a protocol execution, if a party p_r fails to receive a valid message from another party p_s, then we say p_r receives ϕ. If p_r is not a corrupted parties and p_r receives ϕ from p_s, assuming $\delta = 0$, then one of the followings must hold:

1. p_s is a corrupted party while p_r is a fault-free party or a constrained party.
2. p_s is a constrained party while p_r is a fault-free party or a constrained party.
3. p_s is a fault-free party while p_r is a constrained party.

If $\delta > 0$, then it is possible that both p_s and p_r are fault-free parties and yet p_r receives ϕ from p_s.

$\perp$ is a special symbol denoting abortion. In any (sub-)protocol execution, if a party outputs $\perp$, then the party aborts the entire execution at that point. We also assume that a constrained party outputs $\perp$ if it fails to contact the common global clock. Our protocols are designed in such a way that *only* a constrained party will output $\perp$. For clarity, when we refer to an uncorrupted party p_i in our proofs, unless otherwise specified, we implicitly assume that p_i is a fault-free party or a constrained party who has not aborted at that point (i.e., we do not consider a constrained party that has already aborted).

2.2 Definition of secure computation

We define the secure multiparty computation using the ideal/real world paradigm. We assume the function g is defined in a way such that if the input of an party is ϕ, then the evaluation of g does not depend on the input of that party and the corresponding output for that party is $\perp$. We also assume that if the output of a party is not equal to $\perp$, then its output contains the set of parties which input ϕ. As a warm-up, we will start with the case of a *non-adaptive* adversary.

The non-adaptive case

Ideal world: In the ideal world, there is a trusted party (TP) which carries out the evaluation of the function g. The evaluation consists of the following steps:

1. The adversary chooses a set of corrupted parties $\mathcal{P}^f$, modifies their inputs (which can become ϕ) and sends them to the trusted party; the adversary chooses three sets of constrained parties $\mathcal{P}^{c_1}$, $\mathcal{P}^{c_2}$ and $\mathcal{P}^{c_3}$;[4] the trusted party receives the private inputs from parties that are not in $\mathcal{P}^f \cup \mathcal{P}^{c_1}$; the trusted party receives ϕ as the input from the constrained parties in $\mathcal{P}^{c_1}$.
2. The trusted party evaluates g. Let $\mathcal{P}^\phi$ be the set of parties which send ϕ to the trusted party.
3. The adversary receives the outputs of the parties in $\mathcal{P}^f$; parties in $\mathcal{P}^{c_1} \cup \mathcal{P}^{c_2}$ receive $\perp$; other parties receive their outputs (note that the outputs contain the set $\mathcal{P}^\phi$).

Real world: In the real world, the parties execute a protocol Π to evaluate g. Corrupted parties may deviate from the protocol in an arbitrary manner. Messages delivery are controlled by the adversary, subject to the constraints in our communication model.

At the end of the protocol execution, the fault-free parties and the constrained parties output their outputs from Π; the real-world adversary generates an output (which can depend on the information it gathers during the execution of Π).

We say a protocol Π is a secure multiparty computation protocol if the following holds: for every real world adversary $\mathcal{A}$, there exists an ideal world adversary $\mathcal{I}$ with the same set of corrupted parties and same set of constrained parties such that (1) and (2) are indistinguishable:

1. The output of $\mathcal{I}$ and the outputs of fault-free parties and constrained parties in the ideal world.
2. The output of $\mathcal{A}$ and the outputs of fault-free parties and constrained parties in the real world.

The adaptive case The only difference between this case and the non-adaptive case is the definition of the ideal world. In the *ideal world*,

1. The adversary chooses a set of corrupted parties $\mathcal{P}^{f_1}$ (in an adaptive manner), modifies their inputs (which can become ϕ) and sends them to the trusted party; the adversary chooses a set of constrained parties $\mathcal{P}^{c_1}$; the trusted party receives the private inputs from parties that are not in $\mathcal{P}^{f_1} \cup \mathcal{P}^{c_1}$; the trusted party receives ϕ as the input from the constrained parties in $\mathcal{P}^{c_1}$.

[4] a constrained party in $\mathcal{P}^{c_3}$ is not distinguishable from a fault-free party in the ideal world, the set $\mathcal{P}^{c_3}$ is defined due to a subtle technical point.

2. The trusted party evaluates g. Let $\mathcal{P}^{\phi}$ be the set of parties who send ϕ to the trusted party.
3. The adversary receives the outputs of the parties in $\mathcal{P}^{f_1}$; depending on the outputs it received, the adversary can (adaptively) choose to corrupt a new set of parties $\mathcal{P}^{f_2}$ and obtain their outputs; the adversary then chooses two sets of constrained parties $\mathcal{P}^{c_2}$ and $\mathcal{P}^{c_3}$; parties in $\mathcal{P}^{c_1}$ and $\mathcal{P}^{c_2}$ receive $\perp$; other uncorrupted parties receive their outputs.

At the end, the fault-free parties and the constrained parties output what they receive from the trusted party; the ideal-world adversary generates an output.

2.3 Overview

In section 5, we construct a MPC protocol for $n > 4e_f + 3e_c$ which uses broadcast(section 3) and VSS(section 4) as sub-protocols. We note that we assume $\delta = 0$(i.e. a fault-free party can always successfully receive messages from other fault-free parties) in sections 3, 4 and 5. In section 6, we discuss how to extend our results to the case of $\delta < \frac{1}{6}$. In section 7, we conclude and state some open problems.

3 Broadcast

3.1 Definitions

In broadcast, there is a distinguished sender p_s with input v. We can define broadcast using the ideal/real world paradigm by specifying the ideal world as follows:

1. The adversary chooses a set of corrupted parties $\mathcal{P}^f$ and a set of constrained parties $\mathcal{P}^c$.
   - If $p_s \in \mathcal{P}^f$, then the adversary obtains the value v and p_s sends a possibly modified value v' (v' can be equal to ϕ) to the trusted party.
   - If $p_s \in \mathcal{P}^c$, then the adversary sends a flag b to p_s; if b is equal to true, then p_s sends v to the trusted party else p_s sends ϕ.
   - If $p_s \notin \mathcal{P}^f \cap \mathcal{P}^c$, then p_s sends v to the trusted party.
2. The trusted party sends the value it received from p_s to all parties not in $\mathcal{P}^c$; it sends ϕ to all parties in $\mathcal{P}^c$.

However, the above definition is an overkill for our application (as a building block for VSS and MPC). If p_s is a constrained party and it fails to broadcast the message (i.e., p_s receives $b =$ false from the adversary and the honest parties receive ϕ), then the adversary should obtain no knowledge about the message. We need some kind of secret sharing to achieve this in the real world. However, our intention is to construct a VSS protocol using broadcast, not the other way round! To solve this dilemma, we observe that in our applications of broadcast, privacy is not an issue. In more details, what we need are as follows:

- If the sender is corrupted, then all fault-free parties receive the same value v' (v' can be equal to ϕ). A constrained party should receive v' or ϕ.
- If the sender is constrained, then all fault-free parties receive the same value v' (v' has to be equal to v or ϕ). A constrained party should receive v' or ϕ.
- If the sender is fault-free, then all fault-free parties receive the same value $v' = v$. A constrained party should receive v' or ϕ.

It will ease the designing of the VSS protocol if we place a more stringent requirement: if a constrained party does not receive v', then it aborts (i.e. outputting $\perp$).

More formally, we say broadcast is achieved if the followings hold:

- Agreement: If an uncorrupted party outputs $v'(\neq\perp)$[5], then all fault-free parties output v'.
- Correctness: If the sender is uncorrupted and an uncorrupted party outputs $v'(\neq\perp)$, then $v' = v$ or $v' = \phi$. If the sender is fault-free, then all fault-free parties output v.

If the sender is constrained, it is possible that all fault-free parties output ϕ. We assume a constrained sender will abort if it outputs ϕ in the broadcast protocol. We reduce the broadcast problem to the *consensus* problem. In consensus, every party p_i has an input v_i. Consensus is achieved if the followings hold:

- Agreement: All fault-free parties output a common value v. A constrained party either outputs v or $\perp$(abort).
- Persistence: If all fault-free parties have the same input v', then $v = v'$.

For the rest of the section, we focus on the case where the domain of values is restricted to {0,1}. It is easy to see that if we have a broadcast protocol for a single bit, then we can have broadcast protocol for a ℓ-bit string by running the bit-protocol ℓ-times sequentially. For a bit b, we denote its complement by $\bar{b}$.

3.2 Reducing broadcast to consensus

Under our communication model, broadcast cannot be achieved by simply having the sender sending its value to all parties and then running the consensus protocol. The problem is that the sender could be constrained and fault-free parties may not receive the value from the sender. Nevertheless, we show the following:

Lemma 1. *Consensus implies broadcast.*

Proof. We construct a broadcast protocol from any consensus protocol:

1. Sender p_s sends the bit v to all parties. Let b_i be the bit p_i received from p_s. If p_i does not receive anything from the sender, then sets $b_i = 0$.

[5] v' can be equal to ϕ

2. Parties execute the consensus protocol. Each party p_i enters the protocol with input b_i and let b_i^* be the output. If $b_i^* =\perp$, then p_i outputs $\perp$ and aborts.
3. If $v = b_s^*$, then p_s sends 1 to all parties; otherwise, p_s does not send anything.
4. If p_i receives 1 from the sender, then sets $d_i = 1$ else sets $d_i = 0$. Parties execute the consensus protocol again. This time, p_i enters the protocol with input d_i and lets d_i^* be the corresponding output. If $d_i^* = 1$, then p_i outputs b_i^* else if $d_i^* = 0$, then p_i outputs ϕ else p_i outputs $\perp$ ($d_i^* = \phi$ for the last case) .

The consensus protocol is run twice in the construction. Roughly speaking, the first execution establishes a common value among the parties. However, if the sender is constrained, then the established value may be different from v. The second execution is to determine if the sender is "happy" with the established value. If the sender is not happy, then all parties output ϕ. The formal proof proceeds as follows:
Agreement: (i) If an uncorrupted party outputs $b \notin \{\phi, \perp\}$, then by the agreement property of consensus, $d_i^* = 1$ and $b_i^* = b$ for all fault-free parties p_i. Hence all fault-free parties output b. (ii) If an uncorrupted party outputs ϕ, then $d_i^* = 0$ for all fault-free parties p_i. Hence all fault-free parties output ϕ.
Correctness: Consider two cases: (i) a fault-free sender and (ii) a constrained sender. (i) If the sender is fault-free, then all fault-free parties receive v from the sender in step 1. By the persistence property of consensus, all fault-free parties have $b_i^* = v$. Hence all fault-free parties p_i receive 1 from the sender in step 3 and sets $d_i = 1$ in step 4. By the persistence property of consensus again, $d_i^* = 1$. Hence a fault-free party outputs v. (ii) If the sender p_s is constrained, consider two sub-cases: (a) $b_s^* = v$ (b) $b_s^* \neq v$. For case (a), a fault-free party p_i may or may not receive 1 from p_s in step 3 and d_i^* may equal to 0 or 1. However, note that $b_i^* = b_s^* = v$. Therefore, p_i either outputs v or ϕ. For case (b), p_s does not send anything in step 3. Hence all fault-free parties p_i enter the consensus protocol in step 4 with input $d_i = 0$. By the persistence property of consensus, $d_i^* = 0$. Therefore all uncorrupted parties output ϕ.

3.3 Consensus for $n > 3e_f + 2e_c$

Following the principle of Berman et al.[5], the construction of the consensus protocol is done through constructing protocols for weaker consensus variants: weak consensus, graded consensus, king consensus and then consensus. In all these (sub-)protocols, we only need to know the number of fault-free parties $e_{ff} \stackrel{\text{def}}{=} n - e_f - e_c$, but not e_f and e_c; moreover, we only require authenticate (but not secure) point-to-point channels. For all these (sub-)protocols, p_i has an input bit and we denote it as b_i.

Weak Consensus We say weak consensus is achieved if the following two conditions hold:

– Persistence: If all fault-free parties have the same input bit b, then all fault-free parties output b.
– Agreement: If an uncorrupted party outputs $b \in \{0, 1\}$, then all uncorrupted parties output b or 2.

Protocol WConsensus(p_i,b_i,e_{ff})

1. p_i sends b_i to all parties.
2. Let X_i^0 and X_i^1 be the number of 0 and 1 received by p_i respectively.
   If $X_i^0 \geq e_{ff}$, p_i outputs 0, else if $X_i^1 \geq e_{ff}$, p_i outputs 1 else p_i outputs 2.

Lemma 2. *Protocol WConsensus achieves weak consensus for $n > 3e_f + 2e_c$.*

Proof. Persistence: Note that $e_{ff} > \frac{n}{2}$. If all fault-free parties p_i have the same input bit b, then $X_i^b \geq e_{ff}$ and $X_i^{\bar{b}} < e_{ff}$. Hence p_i will output b. *Agreement:* Suppose there exists two uncorrupted parties p_i and p_j outputting 0 and 1 respectively. Then $|X_i^0 \cap X_j^1| \geq e_{ff} - (e_f + e_c) = n - e_f - e_c - (e_f + e_c) > e_f$. Therefore, there exists more than e_f parties sending different bits to p_i and p_j in round 1. This is a contradiction since there are at most e_f corrupted parties.

Graded consensus In graded consensus, every party p_i outputs a bit along with a grade g_i. Graded consensus is achieved if the following three conditions are satisfied:

– Persistence: If all fault-free parties have the same input bit b, then all fault-free parties output b with $g = 1$.
– Agreement: If an uncorrupted party outputs b with $g = 1$, then all fault-free parties output b, all constrained parties output b or $\perp$.
– Completeness: No fault-free party outputs $\perp$.

Protocol GConsensus(p_i,b_i, e_{ff})

1. p_i sends the output of WConsensus(p_i, b_i, e_{ff}) to all parties.
2. Let X_i^0, X_i^1 and X_i^2 be the number of 0, 1 and 2 received by p_i respectively.
   If $\max\{X_i^0, X_i^1\} + X_i^2 < e_{ff}$, then p_i outputs $\perp$ and abort.
   If $X_i^0 \geq e_{ff}$, p_i outputs 0 with $g_i = 1$,
   else if $X_i^1 \geq e_{ff}$, p_i outputs 1 with $g_i = 1$,
   else if $X_i^0 > X_i^1$, p_i outputs 1 with $g_i = 0$,
   else p_i outputs 0 with $g_i = 0$.

Lemma 3. *Protocol GConsensus achieves graded consensus for $n > 3e_f + 2e_c$.*

Proof. Persistence: If all fault-free parties have the same input bit b, then following the persistence property of weak consensus, they output the same bit b in WConsensus. For a fault-free party p_i, $X_i^b \geq e_{ff}$ and $X_i^{\bar{b}} < e_{ff}$. Therefore p_i outputs b with $g_i = 1$. *Agreement:* If an uncorrupted party p_i outputs b with $g_i = 1$, then $X_i^b \geq e_{ff}$. Hence at least $e_{ff} - e_f$ uncorrupted parties have b as the output of WConsensus. Following the agreement property of weak consensus, the number of uncorrupted parties that output $\bar{b}$ in WConsensus is equal to 0. By counting, the number of uncorrupted parties that output 2 in WConsensus is at most $e_{ff}+e_c-(e_{ff}-e_f)=e_c+e_f$. Assume on contrary that there exists an uncorrupted party p_j outputs $\bar{b}$ in GConsensus. Then $X_j^{\bar{b}} \geq X_j^b$. Note that all $\bar{b}$ p_j received in step 1 are from corrupted parties. Therefore, $X_j^2 \leq e_f+e_c+(e_f-X_j^{\bar{b}})$. ($e_f+e_c$ corresponds to the number of 2 received due to uncorrupted parties; $e_f - X_j^{\bar{b}}$ corresponds to the number of 2 received due to corrupted parties.) But $X_j^{\bar{b}} + X_j^2 \leq X_j^{\bar{b}} + e_f + e_c + (e_f - X_j^{\bar{b}}) = 2e_f + e_c < e_{ff}$ as $n > 3e_f + 2e_c$ and $e_{ff} = n - e_f - e_c$. Therefore, $\max\{X_j^0, X_j^1\} + X_j^2 < e_{ff}$, p_j should output $\perp$ instead. Contradiction. *Completeness:* By the agreement property of weak consensus, for some bit b, each fault-free party has either b or 2 as the output of WConsensus. Therefore, for a fault-free party p_i, $\max\{X_i^0, X_i^1\} + X_i^2 \geq e_{ff}$. Hence no fault-free party outputs $\perp$.

King Consensus In king consensus, there is a designated party p_k known as the king. King consensus is achieved if the followings hold:

- Persistence: If all fault-free parties have the same input bit b, then all uncorrupted parties that do not abort output b.
- Correctness: If the king p_k is fault-free, then all uncorrupted parties that do not abort output the same bit.
- Completeness: No fault-free party outputs $\perp$.

Protocol KConsensus$_{p_k}(p_i, b_i, e_{ff})$

1. Let (v_i, g_i) be the output of GConsensus(p_i, b_i, e_{ff}). If $(v_i, g_i) = \perp$, then p_i outputs $\perp$.
2. p_k sends v_k to all parties.
3. If $(g_i \neq 1)$ and p_i receives v_k from p_k and $v_k \neq \phi$, then p_i sets $v_i = v_k$.
4. Let (v_i', g_i') be the output of GConsensus(p_i, v_i, e_{ff}). p_i outputs v_i'.

Lemma 4. *Protocol KConsensus achieves king consensus for $n > 3e_f + 2e_c$.*

Proof. Persistence: If all fault-free parties have the same input bit b, then by the persistence property of graded consensus, all fault-free parties p_i have $(b, 1)$ as the output of the first execution of GConsensus, i.e., $v_i = b$ and $g_i = 1$. Since $g_i = 1$, v_i will not be modified in step 3. All fault-free parties enter the second execution of GConsensus with the same input b. By the persistence and the agreement properties of graded consensus, all uncorrupted parties output the bit b in KConsensus. *Correctness:* Suppose p_k is a fault-free party. Consider two cases: (i) there exists a fault-free party p_i with $g_i = 1$ by the end of step 1. (ii) all fault-free parties p_i have $g_i = 0$ by the end of step 1. For case (i), following the agreement property of graded consensus, all fault-free parties p_j (including p_k) have the same value for v_j (i.e., $v_j = v_i = v_k$) by the end of step 1. It does not matter whether p_j resets its value in step 3. For case (ii), all fault-free parties p_j receive v_k from p_k in step 3 and set $v_j = v_k$. Combining two cases, all fault-free parties enter the second execution of GConsensus with the same input v_k. Following the persistence and agreement properties of graded consensus, all uncorrupted parties output v_k in KConsensus. *Completeness:* Completeness of KConsensus follows the completeness of graded consensus since no fault-free party will output $\perp$ in the executions of GConsensus.

Consensus We show how to construct a consensus protocol from a king consensus protocol:

<u>Procotol Consensus(p_i, b_i, e_{ff})</u>

1. Set $b'_i = b_i$.
2. for $k = 1$ to $n - e_{ff} + 1$ do:
   (a) Set b'_i to the output of $\mathrm{KConsensus}_{p_k}(p_i, b'_i, e_{ff})$.
   (b) If $b'_i = \perp$, then p_i outputs $\perp$ and abort.
3. p_i outputs b'_i.

Theorem 1. *Protocol Consensus achieves consensus for $n > 3e_f + 2e_c$.*

Proof. Persistence: If all fault-free parties enter the protocol Consenus with the same input bit b, then by the persistence property of king consensus, all uncorrupted parties that do not abort output the same bit b. *Agreement:* Note that $n - e_{ff} + 1 = e_f + e_c + 1$. There exists a fault-free party $p_i \in \{p_1, \ldots, p_{e_f+e_c+1}\}$. By the correctness property of king consensus, all fault-free parties will have the same value for b' after $\mathrm{KConsensus}_{p_i}$ is run. Agreement then follows from the persistence property of king consensus.

3.4 Consensus for $n \geq 3e_f + 2e_c$, $e_f, e_c \geq 1$

If the values of e_f and e_c are known a priori, then we can improve the bound in Theorem 1. On a high level, the construction takes two steps. First, based on the consensus protocol we have for $n > 3e_f + 2e_c$, we construct a *weak broadcast* (to be defined) protocol for $n \geq 3e_f + 2e_c$. Second, we convert a weak broadcast protocol into a consensus protocol.

Weak Broadcast In weak broadcast, there is a sender p_s with an input bit b. Weak broadcast is achieved if the following two conditions hold:

– Agreement: All fault-free parties output a common bit b'.
– Correctness: If the sender is fault-free, then $b = b'$.

Note that in weak broadcast, we do not concern the outputs of constrained parties. Due to lack of the space, we omit the description of the protocol. The details will appear in the full version [6].

[6] A preliminary full version is available at the author's homepage: http://www.cs.umd.edu/∼cykoo

From weak broadcast to consensus Once we have a protocol for weak broadcast, it is easy to construct a consensus protocol:

1. Each party p_i weak-broadcasts the input bit b_i using the protocol WBroadcast.
2. If the majority of the broadcasted bits is 1, then p_i sets $b'_i = 1$ else p_i sets $b'_i = 0$. p_i sends b'_i to all parties.
3. Let X_i^0 and X_i^1 be the number of 0 and 1 received by p_i in last round respectively. If $X_i^0 > \frac{1}{2}n$, then p_i outputs 0 else if $X_i^1 > \frac{1}{2}n$, then p_i outputs 1 else p_i outputs $\perp$.

Since all fault-free parties p_i have the same output in weak broadcasts, they will have the same value for b'_i. In particular, if all of them have the same input bit b, then $b'_i = b$. As the majority of parties are fault-free, if an uncorrupted party p_j receives $> \frac{1}{2}$ copies of $b'_j \in \{0, 1\}$ in round 2, then $b'_j = b'_i$ for any fault-free party p_i. Therefore both persistence and agreement properties hold. Hence we have the following:

Theorem 2. *There is a consensus protocol for $n \geq 3e_f + 2e_c$, $e_f \geq 1, e_c \geq 1$, assuming the values of e_f and e_c are known a priori.*

4 Verifiable secret sharing (VSS)

In verifiable secret sharing (VSS), there is a special party p_d known as the dealer. The dealer holds a secret s. A VSS protocol consists of two phases: a *sharing phase* and a *reconstruction phase*. In the sharing phase, the dealer shares the secret with other parties. Parties may *disqualify* a non fault-free dealer. If the dealer is not disqualified, then in the reconstruction phase, the parties reconstruct a value based on their views in the sharing phase.

In our case, VSS protocol is used as a tool for multiparty computation. Our definition requires a VSS protocol to have the verifiable secret and polynomial sharing property[15]. In this section, we assume the values of e_f and e_c are known a priori. We say a protocol achieves *verifiable secret sharing* if the followings hold:

- Privacy: If the dealer is uncorrupted, then the view of the adversary during the sharing phase reveals no information on s.
- Agreement: If an uncorrupted party disqualifies the dealer, then all uncorrupted parties that do not abort disqualify the dealer.
- Commitment: If the dealer is not disqualified, then there exists a polynomial $h'(x)$ of degree e_f such that at the end of the sharing phase, all fault-free parties p_i (locally) output $h'(i)$; a constrained party p_j which does not abort outputs $h'(j)$. All uncorrupted parties that do not abort output $h'(0)$ in the reconstruction phase.
- Correctness: No fault-free party will abort the protocol. A fault-free dealer will not be disqualified while a constrained dealer may be disqualified. But if an uncorrupted dealer is not disqualified, then $h'(0) = s$.

Theorem 3. *Assuming the values e_f and e_c are known a priori, there exists a VSS protocol for $n \geq 4e_f + 3e_c + 1$.*

Proof. We construct a VSS protocol with the above resilience. The protocol is based on the bivariate solution of Feldman-Micali[13]. We start by giving a high level description of the protocol. In round 1, the dealer shares the secret via a random bivariate polynomial of degree $e_f + 1$. If the dealer is constrained, then a fault-free party may not receive its entitled share. However, unlike [13], the fault-free party cannot take a default value since it will not be on the polynomial and correctness will be violated. Instead, a party broadcasts "receive" in round 2 if it has received its entitled share. Let $\mathcal{G}$ be the group of parties who broadcast "receive". If $|\mathcal{G}|$ is too small, then the dealer is disqualified. Otherwise, the parties within $\mathcal{G}$ proceed to verify if the dealer has shared a valid secret, using a similar approach as in [13](with suitable modifications to tolerate the presence of constrained parties). The parties that are not in $\mathcal{G}$ will not take part in the verification. After the verification, if the dealer is not disqualified, then all parties in $\mathcal{G}$ have shares correspond to a valid secret. The parties outside $\mathcal{G}$ then compute their shares by interpolating the shares from the parties in $\mathcal{G}$ (here we exploit the fact that the secret is shared using a bivariate polynomial).

We assume the secret s is taken from some finite field $\mathcal{F}$. In the following, if the dealer broadcasts a value and the parties receive ϕ as the output, then we implicitly assume that all parties disqualify the dealer.

VSS-share(p_d)

1. The dealer chooses a random bivariate polynomial f of degree at most e_f in each variable such that $f(0,0) = s$. The dealer sends to party p_i the polynomials $g_i(x) \stackrel{\text{def}}{=} f(x,i)$ and $h_i(x) \stackrel{\text{def}}{=} f(i,x)$.
2. p_i broadcasts "0" if it does not receive $g_i(x)$ and $h_i(x)$ from the dealer (or $g_i(x)$ and $h_i(x)$ are not polynomials of degree e_f), otherwise p_i broadcasts "1". Let $\mathcal{G}$ be the group of parties which broadcast "1". If $|\mathcal{G}| < 3e_f + 2e_c + 1$, then the dealer is disqualified. Otherwise, each party $p_i \in \mathcal{G}$ does the following:
   (a) For every party $p_j \in \mathcal{G}$, p_i sends $g_i(j)$ and $h_i(j)$ to p_j. Let $g'_{j,i}$ and $h'_{j,i}$ be the two values received by p_i from $p_j \in \mathcal{G}$. p_i aborts if it receives values from less than $2e_f + e_c + 1$ parties.
   (b) For every party $p_j \in \mathcal{G}$, if ($g'_{j,i} \neq h_i(j)$ and $g'_{j,i} \neq \phi$) or ($h'_{j,i} \neq g_i(j)$ and $h'_{j,i} \neq \phi$), then p_i broadcasts "complaint : i,j" else p_i broadcasts "no complaint: i,j". Note that if an uncorrupted p_i broadcasts "complaint: i,j", then p_j or the dealer is corrupted.
   (c) The dealer broadcasts $f_{j,k} \stackrel{\text{def}}{=} f(j,k)$ and $f_{k,j} \stackrel{\text{def}}{=} f(k,j)$ if "complaint: j,k" is broadcasted by a party $p_j \in \mathcal{G}$.
   (d) If there exists a j such that (i) $f_{j,i}$ and $f_{i,j}$ are revealed in last step and (ii) $f_{j,i} \neq g_i(j)$ or $f_{i,j} \neq h_i(j)$, then p_i broadcasts "complaint", otherwise p_i broadcasts "okay". (If an uncorrupted p_i broadcasts "complaint", then the dealer must be corrupted. On the other hand, if the dealer is uncorrupted and p_i broadcasts "complaint", then p_i must be corrupted.)

(e) If p_i broadcasts "complaint" in last step, then the dealer broadcasts $g_i(x)$ and $h_i(x)$.

(f) p_i broadcasts "reject" if one of the followings hold:
   - p_i broadcasts "complaint" in step 2(d)
   - There exists a public polynomial $g_k(x)$ and $h_k(x)$ such that $g_k(i) \neq h_i(k)$ or $h_k(i) \neq g_i(k)$
   - The dealer does not respond to the complaints broadcasted in step 2(b) or step 2(d);

   otherwise, p_i broadcasts "accept".

3. If less than $3e_f + 2e_c + 1$ parties in $\mathcal{G}$ broadcast "accept", then the dealer is disqualified. Otherwise, note that two polynomials $g_i(x)$ and $h_i(x)$ are associated with each uncorrupted party p_i in $\mathcal{G}$ (If the polynomials are not made public in step 2(e), then the polynomials associated with p_i are the two polynomials p_i received in step (1)). Each party $p_i \in \mathcal{G}$ sends $h_i(j)$ to all parties $p_j \notin \mathcal{G}$.
4. For each party $p_i \notin \mathcal{G}$, p_i constructs a degree e_f polynomial $g_i(x)$ by using the Reed-Solomon error-correction interpolation procedure on the values it received in last step (If p_i cannot construct such polynomial or p_i receives less than $2e_f + e_c + 1$ shares, then p_i aborts).
5. Each party p_i outputs $g_i(0)$.

VSS-reconstruct($g_i(0)$)

1. Party p_i sends $g_i(0)$ to all parties.
2. Let SS^i be the set of secret shares p_i receives in last round. If $|SS^i| < 3e_f+1$, then p_i aborts else p_i reconstructs a polynomial $h_0(x)$ of degree e_f by using the Reed-Solomon error-correction interpolation on the set SS^i and outputs $h_0(0)$.

We now proceed to prove that the above protocol achieves VSS.

- Privacy: Consider an uncorrupted dealer. If an uncorrupted party p_i broadcasts "complaint: i,j" in step 2(b), then p_j must be a corrupted party. It is easy to see that if a party p_i broadcasts a complaint in step 2(d), then p_i is a corrupted party. Therefore, all the information broadcasted by an uncorrupted dealer on f, if any, is a subset of the shares the corrupted parties entitled to receive in step 1. Since the secret is shared by a random bivariate polynomial of degree e_f, we conclude that the view of the adversary is independent of s during the sharing phase.
- Agreement: Note that the decision of disqualifying a dealer is completely dependent on the messages broadcasted by the parties. If an uncorrupted party does not abort by the end of the sharing phase, then by the agreement property of broadcast, the values it received from the broadcasts are same as those received by fault-free parties. Hence agreement follows.
- Correctness: We first consider a fault-free dealer. All fault-free parties will be in $\mathcal{G}$. Since $n \geq 4e_f + 3e_c + 1$, it follows that $|\mathcal{G}| \geq 3e_f + 2e_c + 1$. In addition, all fault-free parties broadcast "accept" in step 2(f). For a constrained party

p_i that is not in $\mathcal{G}$, it is easy to see that $g_i(x)$ reconstructed in step 3 (if p_i does not abort) is equal to $f(x,i)$.

Next we consider a constrained dealer that is not disqualified. For an uncorrupted party $p_i \in \mathcal{G}$ that does not abort by step 2(f), it is easy to see that $g_i(x) = f(x,i)$ and $h_i(x) = f(x,i)$. If the dealer is not disqualified, then $\geq 3e_f + 2e_c + 1$ parties broadcasts "accept" in step 2(f). Let $\mathcal{G}'$ be the set of fault-free parties among these $\geq 3e_f + 2e_c + 1$ parties. $|\mathcal{G}'| \geq 2e_f + e_c + 1$. It then follows that every fault-free party (or constrained party that does not abort in step 3) p_i that is not in $\mathcal{G}$ can reconstruct $g_i(x) = f(x,i)$. An uncorrupted party p_i (if it does not abort) will then output $f(0,i)$ in step 4.

– Commitment: We consider the case of a non-disqualified corrupted dealer. If a corrupted dealer is not disqualified, then at least $2e_f + e_c + 1$ fault-free parties broadcast "accept" in step 2(f). Let $\mathcal{G}'$ be the set of such fault-free parties. Following [13, Lemma 2], there exists a bivariate polynomial f' of degree e_f in each variable such that for all $p_i \in \mathcal{G}'$, $g_i(x) = f'(x,i)$ and $h_i(x) = f'(i,x)$. Now consider an uncorrupted party $p_j \in \mathcal{G}$ but not in $\mathcal{G}'$. There are 2 possible scenarios:
  - p_j broadcasts a complaint in step 2(d). $g_j(x)$ and $h_j(x)$ are made public in step 2(e). For all $p_i \in \mathcal{G}'$, $g_j(i) = h_i(j)$ and $h_j(i) = g_i(j)$. Hence it follows that $g_j(x) = f'(x,j)$ and $h_i(x) = f'(x,i)$.
  - p_j does not broadcast a complaint in step 2(d). If p_j does not abort in step 2(b), then $h_j(i) = f_i(j)$ and $f_j(i) = h_i(j)$ for at least $2e_f + e_c + 1 - (e_f + e_c) = e_f + 1$ parties $p_i \in \mathcal{G}'$. Since f' is a bivariate polynomial of degree e_f, it follows that $h_j(x) = f'(j,x)$ and $g_j(x) = f'(x,j)$. Therefore if p_j does not broadcast a complaint in step 2(d), it will not broadcast "reject" in step 2(f).

  We conclude that for all uncorrupted parties $p_i \in \mathcal{G}$ that do not abort by step 2, $h_i(x) = f'(i,x)$ and $g_i(x) = f'(x,i)$. It is easy to see that if an uncorrupted party $p_j \notin \mathcal{G}$ does not abort by step 3, p_j can reconstruct $g_j(x) = f'(x,j)$. Hence it follows that an uncorrupted party p_i (if it does not abort) outputs $f'(0,i)$ in step 4.

  It also follows that all uncorrupted parties that do not abort output $h'(0)$ by the end of VSS-reconstruct.

5 MPC

We now construct a MPC protocol following the paradigm in [3]. On a high level, each party shares its private input, evaluates the circuit gate by gate, and then reconstructs the outputs.

Input Phase: Every party shares its private input using VSS-share. If a party is disqualified (when it plays the role of the dealer), then the party is added to the set $\mathcal{D}$. Note that by the end of the input phase, all uncorrupted parties that do not abort have the same view on $\mathcal{D}$.

Circuit Evaluation: All parties that do not abort evaluate the circuit $g^{\mathcal{D}}$ gate by gate. A party who was disqualified in the input phase does not take part in this phase and all other parties will ignore the messages sent from that party. It suffices to consider the addition and multiplication gates. The evaluation procedures are very similar to the one in [7, section 4.52] and we omit the details here.

Output phase: Reconstructing output is easy. For each output wire, each party sends its share to the party who is entitled to receive the output. The corresponding party then reconstructs the output from the shares it received using error correction. A party aborts if it receives less than $3e_f + 1$ entitled shares.

6 Extending to the case of $\delta < \frac{1}{6}$

We describe how to extend the results from the previous sections (which assume $\delta = 0$) to the case of $\delta < \frac{1}{6}$, at the expense of increasing the round complexity by a factor of 2. More precisely, we show how to compile a protocol Π for $\delta = 0$ into a protocol Π' for $\delta < \frac{1}{6}$.

Our broadcast protocol Π for $\delta = 0$ assumes $n \geq 3e_f + 2e_c$ but does not assume secure channels. If p_i is supposed to send p_j a message m in Π, then the followings are carried out in Π':

- p_i sends m to all parties who then forward the message to p_j.
- If there exists m' such that p_j receives $\geq \frac{4}{3}e_f$ copies of them, then p_j sets $m = m'$ else $m = \phi$.

Consider the following two cases:

1. Both p_i and p_j are fault-free parties: since $n \geq 3e_f + 2e_c$, at least $(2e_f + e_c)(1 - \delta)$ fault-free parties receive m from p_i. The number of copies of m p_j received is at least $(2e_f + e_c)(1 - 2\delta)$ which is greater than $\frac{4}{3}e_f$ if $\delta < \frac{1}{6}$. Hence p_j can receive m from p_i.
2. At least one of the p_i and p_j is a constrained party: suppose p_j receives m' from p_i in Π' and $m' \neq \phi$. Since p_j receives at least $\frac{4}{3}e_f$ copies of m', at least $\frac{1}{3}e_f$ copies are from uncorrupted parties. Hence $m' = m$ (assuming $e_f \geq 3$).

For VSS and MPC protocols, we assume $n \geq 4e_f + 3e_c + 1$ but we also assume secure channels. If p_i is supposed to send p_j a message m in Π, then the following steps are carried out in Π':

- p_i picks a random polynomial $h(x)$ of degree e_f such that $h(0) = m$. p_i sends $h(k)$ to p_k who then forwards the share to p_j.
- Based on the shares p_j received, using the Reed-Solomon error-correction interpolation procedure, p_j constructs a polynomial $h'(x)$ of degree e_f such that at least $2e_f + 1$ shares are on $h'(x)$. If p_j cannot construct such polynomial, then p_j sets $m = \phi$ else $m = h'(0)$.

First we note that if p_i is uncorrupted, then the view of the adversary is independent of m since h is a random polynomial of degree e_f. Second, if both p_i and p_j are uncorrupted and p_j does not set $m = \phi$, then p_j receives $2e_f + 1$ shares that are on $h'(x)$. $e_f + 1$ of these shares are from uncorrupted parties. Hence $h'(x) = h(x)$ since both $h'(x)$ and $h(x)$ are of degree $e_f + 1$. Finally, if both p_i and p_j are fault-free parties, then p_j will receive at least $(3e_f+2e_c+1)(1-2\delta) \geq 2e_f + 1$ (assuming $\delta < \frac{1}{6}$ and $e_c \geq 1$) correct shares. On the other hand, p_j will receive at most e_f corrupted shares. Hence p_j can always reconstruct $h(x)$ using error-correction.

7 Conclusion and Open problems

In this paper, we consider a communication model where message delivery is neither always guaranteed nor always in the hands of the adversary. We has developed broadcast and VSS protocols under this model. However, we do not know if the bounds are tight. Another interesting direction is to consider what is achievable if the global clock is removed from the model.

Acknowledgments

The author thanks Omer Horvitz, Jonathan Katz, Ruggero Morselli, Tsuen-Wan "Johnny" Ngan and Ji Sun Shin for helpful comments and encouragement. In particular, discussions on the communication model with Ruggero Morselli and Tsuen-Wan "Johnny" Ngan are very helpful. The author also thanks the anonymous referees for providing useful references and thoughtful comments.

References

1. D. Beaver. Multiparty protocols tolerating half faulty processors. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference*, volume 435 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 1989.
2. M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 52–61, New York, NY, USA, 1993. ACM Press.
3. M. Ben-Or, S. Goldwasser, and A.Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computations. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, pages 1–10, 1988.
4. M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *PODC '94: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 1994. ACM Press.
5. P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 410–415. IEEE, 1989.

6. M. Biely. Optimal agreement protocol in malicious faulty processors and faulty links. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):266–280, 1992.
7. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136, Washington, DC, USA, 2001. IEEE Computer Society.
9. R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 15–24, New York, NY, USA, 1997. ACM Press.
10. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 494–503, New York, NY, USA, 2002. ACM Press.
11. D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM Press.
12. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer-Verlag, May 1999.
13. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
14. J. A. Garay and K. J. Perry. A continuum of failure models for distributed computing. In *Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 153–165. Springer-Verlag, 1992. Full version availabe at http://cm.bell-labs.com/who/garay/continuum.ps.
15. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111, New York, NY, USA, 1998. ACM Press.
16. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229. ACM Press, 1987.
17. S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC '02: Proceedings of the 16th International Conference on Distributed Computing*, pages 17–32, London, UK, 2002. Springer-Verlag.
18. Y. T. Kalai, Y. Lindell, and M. Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 644–653, New York, NY, USA, 2005. ACM Press.
19. A. D. Keromytis, V. Misra, and D. Rubenstein. Sos: secure overlay services. *SIGCOMM Comput. Commun. Rev.*, 32(4):61–72, 2002.
20. P. R. Parvédy and M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 302–310. ACM Press, 2004.
21. K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Softw. Eng.*, 12(3):477–482, 1986.

22. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, New York, NY, USA, 1989. ACM Press.
23. U. Schmid, B. Weiss, and J. Rushby. Formally verified byzantine agreement in presence of link faults. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 608, Washington, DC, USA, 2002. IEEE Computer Society.
24. P. Thambidurai and Y.-K. Park. Interactive consistency with multiple failure modes. In *Proceedings of the 7th Reliable Distributed Systems Symposium*, pages 93–100, 1988.
25. K. Q. Yan, Y. H. Chin, and S. C. Wang. Optimal agreement protocol in malicious faulty processors and faulty links. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):266–280, 1992.
26. A. C.-C. Yao. Protocols for secure computations. In *FOCS '82: Proceedings of the 23rd Symposium on Foundations of Computer Science*, pages 160–164, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.