An Optimal Monte Carlo Type Byzantine Agreement Protocol

Aristeidis Tentes

National Technical University of Athens
aristent@hotmail.com

Abstract

The present paper deals with the problem of Byzantine Agreement in a synchronous network in the presence of a polynomially bounded and malicious adversary. There are n processors, which want to agree on a common value, but t of them might be corrupted by an adversary, who tries to make the protocol fail. We present a Monte Carlo type protocol, which is optimal in fault tolerance ($n/2-1$ faults may occur) and in Round Complexity ($O(1)$). Moreover, the Bit Complexity of this protocol is $O(n^2 |\text{ó}|)$ (where $|\text{ó}|$ the length of a signature), which is the lowest achieved until now for such a protocol. Until now the most efficient Monte Carlo protocols either required $O(n^3 |\text{ó}|)$ bits during interaction or tolerated at most $n/3-1$ corrupted players.

Key Words: Cryptography, Byzantine Agreement, Threshold Signature, Common Coin

1. Introduction

The problem of Byzantine Agreement was introduced by Pease, Shostak and Lamport in.[PSL80] It deals with the problem of Consensus, where n processors want to agree on a common value, despite the presence of an adversary, who corrupts t of them and wants to confuse them. An equivalent alternative is the problem of Broadcast, where a processor wants to distribute his value to n processors, but the latter want to ensure that they all received the same value. Many protocols have been proposed since then providing different kinds of security.

It has been proved that any deterministic protocol can tolerate at most $t < n/3$ bad players and requires at least $t+1$ rounds. The most known protocols for this setting are those proposed in [BGP92] and [CW92], where the total number of bits needed during interaction is optimal, namely $O(n^2)$. The protocol of [FM88] is a Las Vegas protocol, which may run in a constant expected number of rounds, but it is also resilient against $t < n/3$ players and requires $O(n^6)$ bits of communication. In the cryptographic setting, the Consensus protocol proposed in [DS83] is deterministic and, although it is optimal in resiliency ($t < n/2$), it requires $t+1$ rounds and a Bit Complexity of $O(n^3 |ó|)$ (where $|ó|$ the length of a digital signature) and the latter is not optimal. If we want a protocol resilient against more than $n/3-1$ corrupted players, then the lower bound for Bit Complexity is $O(n^2 |ó|)$, which also implies that a signature scheme is compulsory.

The Consensus protocol proposed here belongs to the cryptographic model and assumes that the adversary is polynomially bounded. It is a Monte Carlo type protocol, because it allows a negligible probability of error. The protocols of [Rab83] and [Tou84] work also for the same model, however, the former is less resilient, while the latter requires a higher Bit Complexity. The present protocol is optimal in resiliency, tolerating up to $t < n/2$ players and runs in a constant number of rounds. It also requires a in bit complexity of $O(n^2 |ó|)$, which means that it is faster and less demanding than the previous protocols of the cryptographic Monte Carlo setting.

The protocol presented here uses techniques from [CKS00], namely the Common Coin protocol and the threshold signature scheme from [Sho00]. (The protocol of [CKS00] works for an asynchronous network, while our protocol is designed for synchronous networks). As a consequence, our protocol requires a Third Trusted Party once and for all, in contrast to those in [Rab83] and [Tou84], which require a trusted dealer to predistribute some data for each implementation.

2. Preliminaries

2.1 Byzantine Agreement

Let us see, now, the exact definition of Byzantine Agreement in both cases of Consensus and Broadcast. In the literature, the term Byzantine General is sometimes used for the case of Broadcast (there is a general, who wants to send a message to all his lieutenants, but the latter want to be sure, that they all receive the same message), therefore we use the term Byzantine Agreement only for the case of Consensus:

Definition 1 *(Broadcast:) Let $P = (P_1, \ldots, P_n)$ be a set of n processors, D be a finite domain and $P_s \in P$ be the sender. Then we say, that Ð is a Byzantine General protocol among processors in P with values in D, where P_s has as input a value $x_s \in D$ and all players finally decide on a value $y_i \in D$, if it satisfies the following conditions:*

- *Validity:* If the sender is honest, then all correct players will decide on x_s;
- *Consistency:* If P_i is a correct player, who decided on value y_i, and, if P_j

is also a correct player, then he also will decide on $y_j = y_i$.

• *Termination:* All players will eventually terminate the protocol;

Definition 2 *(Consensus:) Let $P = (P_1, \ldots, P_n)$ be a set of n processors, D be a finite domain. Then we say, that Đ is a Byzantine Agreement protocol among processors in P with values in D, where each P_i begins with an input value $x_i \in D$ and finally decides on a value $y_i \in D$, if it satisfies the following conditions:*

• *Validity or Persistency:* If all honest players have as input the same value x, then all honest players will decide on x;

• *Consistency:* If P_i is a honest player, who decided on value y_i, and, if P_j is also a honest player, then he will also decide on $y_j = y_i$.

• *Termination:* All players will eventually terminate the protocol;

In the above definitions, we say that the message space is a (finite) set D, but we can assume, without loss of generality, that $D = \{0,1\}$. By encoding each element of a finite set, with a binary string and running a protocol with $D = \{0,1\}$ many times, we may restrict our attention only to the binary case.

In Byzantine Agreement protocols, there are three parameters, which we want to optimize:

Bit Complexity This is the total number of bits sent by all players. It is denoted by BC.

Round Complexity This is denoted by RC and means the total time, which the protocol requires, until it achieves Byzantine Agreement (or General).

Resiliency It means the number of corrupted players, up to which a protocol can tolerate. If a protocol can handle up to $t < an$ actively corrupted players, then we say that it is a-resilient.

There are two different kinds of Byzantine Agreement protocols. There are protocols, which satisfy the conditions in the absolute sense, without making any assumptions for the honest nor the corrupted players (the adversary). We say that these protocols provide information-theoretic security. On the other hand there are protocols, which assume that there is some consistently shared data (e.g. PKI), before the beginning of the protocol. The most common case is that these protocols also assume that the adversary's computational power is not infinite, but it is polynomially bounded. As we have mentioned our protocol works for the latter case, it is based, namely, on some cryptographic assumptions, namely the discrete logarithm and the Diffie-Hellman problem.

3. Basic Tools

In this section we describe the basic tools, which are used in our protocol, namely the Threshold Signature Scheme and the Common Coin protocol. The full protocols can be found in [Sho00] and in [CKS00] respectively.

3.1 Threshold Signatures

The basic idea behind *Threshold Signatures* is the following: Assume that there is a message given to a group of people. Some of them sign this message and some not. If at leat k of them signed this message, then we say that this group signed this message. An obvious way to check if the group signed the message is to collect at least k different signed messages (e.g. using RSA signature scheme [RSA78]) and to verify each signature one by one. Using a *Threshold Signatures* scheme, though, the group by itself can produce a single signature on this message and anyone, who verifies this signature (he collects only one signature and not more) is sure that at least k of them signed the message. Note we are not concerned to prove, which of them did sign the message but that at least any k of them.

Here is an overview of a (n,k,t) *Threshold Signature* scheme. There is a set of n parties, where t of them may be corrupted and k of them are both necessary and sufficient to produce a *Threshold Signature* on a message.It is obvious that $t < k \leq n-t$ must hold in order our scheme to be meaningful. We have to mention that signatures are made by parties which belong to this set of n, but can be verified by anyone.

The dealer generates a public key PK, a private secret key for each party SK_i, where all these private keys are shares of a specific secret value. In addition the dealer generates a global verification key VK and a local verification key for each player VK_i. Each player P_i possesses his private key SK_i, public key PK and all verification keys. Suppose that a set of k players are asked to produce a threshold signature on a message. Then every player of this set produces a signature share on this message using his private key, sends it to the others. After collecting the other $k-1$ signature shares he checks with a $\mathit{Share\,Verification}$ algorithm their validity. If they all are valid then he uses a $\mathit{Share\,Combining}$ algorithm to produce a threshold signature on the message. This signature can be verified by anyone using the public key PK. As long as $k > t$, it is obvious that if someone checks a threshold signature on a message and computes that it is valid, then all k parties have signed the message. We have described everything we need to define a (n,k,t) *Threshold Signature* scheme:

Definition 3 *A* (n,k,t) *Threshold Signature scheme consists of the following three parts:*

- **Dealer's action**: The dealer generates a public key PK, n different key shares $SK_1,\ldots,SK_N$ and n different local verification keys $VK_1,\ldots,VK_N$. Then he sends to all players the public key, all verification keys and to player P_i the secret key SK_i. The verification key VK_i corresponds to player P_i.

• **Signature Verification Algorithm**: It verifies the validity of a threshold signature, which was produced by the Share Combining Algorithm.

• **Share Verification Algorithm**: It verifies that a share signature, which P_i claims to have properly produced, is valid. To do this it uses the public key PK, the verification key VK and the local verification key of P_i, namely VK_i.

• **Share Combining Algorithm**: It produces the threshold signature using k valid signature shares on the message. It also uses the public key PK and the verification keys.

We say that a *Threshold Signatures* scheme is secure iff the following requirements hold:

$Robustness$: It is computationally infeasible for the adversary to produce k valid signature shares such that the output of the $sharecombining$ algorithm is a valid signature.

$Non-forgeability$: It is computationally infeasible for the adversary to produce a signature on a message if the uncorrupted players who produced a signature share are less than $k-t$.

3.2 Common Coin

There are n parties, which want to produce a random coin value (0 or 1) in such a way that if the adversary corrupts up to t of them, he cannot predict the the value of it nor influence the output. In other words his prediction of the coin value is $1/2$ and, furthermore, the security of the scheme is the same as if all n parties were together and tossed an unbiased coin. We say that we have a (n,k,t) $CommonCoin$ scheme, if k players are both necessary and sufficient to produce a common coin value.

The techniques used in such a scheme are almost the same as in *Threshold Signature* protocol. There is a dealer, who generates secret keys $SK_1,\ldots,SK_n$, one for each player. Then, he computes the local verification keys $VK_1,\ldots,VK_n$, one for each player, which are dependent from the secret keys. After the dealing phase each player P_i possesses all verification keys along with his own secret key.

Using these values each player computes a share of the coin and then he collects k valid coin shares and combines them to compute the value of the coin. We denote by C the name of the coin and by $F(C)$ its value.

Definition 4 *The security requirements of a* (n,k,t) *Common Coin protocol, which produces a common value between 0 and 1 for* n *players are the following:*

$Robustness$: It is computationally infeasible for the adversary to produce a name C and k valid coin shares of C such that the output, when combining the shares is not $F(C)$.

$Unpredictability$: The probability that the adversary may predict the correct output as value of the coin C is negligibly bigger than $1/2$.

4. Byzantine Agreement Protocol

Now, we are ready to present the Byzantine Agreement protocol. In the protocol we mention $BANr$ (Byzantine Agreement Number),which is a counter, which increases every time the same set of parties uses this protocol. The reason is that the signature shares must be valid only for one implementation of the protocol, so that the adversary will not be able to keep some messages and use them another time. Moreover, we use a $(n,n-t,t)$ Threshold Signature scheme (and this is what it meant when a player is asked to produce Thr-signature, namely a $(n,n-t,t)$ threshold signature) and a $(n,n-t,t)$ Coin - Tossing scheme. This is the reason, why this protocol requires a trusted third party in the set-up phase, but for the same set of parties this precomputation phase is needed only once and thereafter they can run the protocol any time they want.

Byzantine Agreement protocol Repeat the following steps for phases $p = 0, 1, \ldots, k$

1. $\forall P_i$ denote by V_i the input of player P_i. Then produces an S-signature share on the message

$$(BANr, p, voteI, V_i)$$

   and send

$$(p, voteI, V_i, signature\ share)$$

   to all the other parties

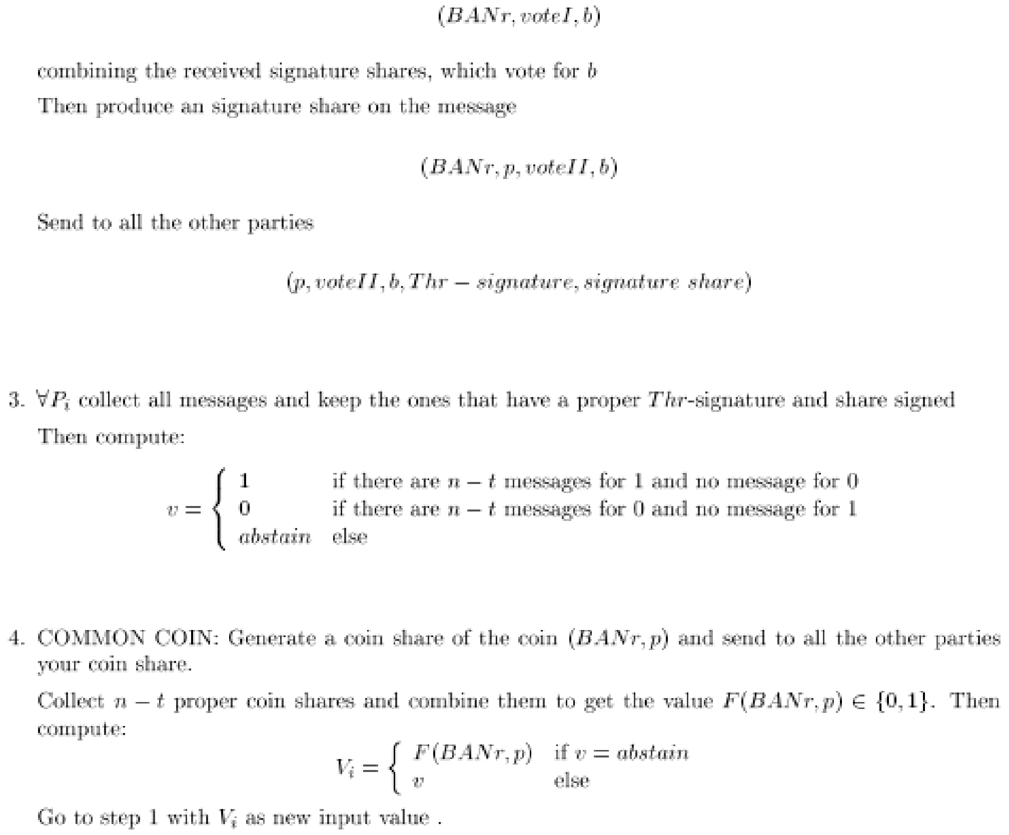
2. $\forall P_i$ collect all voteI messages and keep the ones which are properly share signed.

   Then compute:

$$b = \begin{cases} 1 & \text{if there are at least } t+1 \text{ votes for } 1 \\ 0 & \text{if there are at least } t+1 \text{ votes for } 0 \\ abstain & \text{else} \end{cases}$$

   Produce an Thr-signature of the message

$$(BANr, voteI, b)$$



$$(BANr, voteI, b)$$

combining the received signature shares, which vote for b
Then produce an signature share on the message

$$(BANr, p, voteII, b)$$

Send to all the other parties

$$(p, voteII, b, Thr - signature, signature\ share)$$

3. $\forall P_i$ collect all messages and keep the ones that have a proper Thr-signature and share signed
   Then compute:

$$v = \begin{cases} 1 & \text{if there are } n-t \text{ messages for 1 and no message for 0} \\ 0 & \text{if there are } n-t \text{ messages for 0 and no message for 1} \\ abstain & \text{else} \end{cases}$$

4. COMMON COIN: Generate a coin share of the coin $(BANr, p)$ and send to all the other parties your coin share.
   Collect $n-t$ proper coin shares and combine them to get the value $F(BANr, p) \in \{0, 1\}$. Then compute:

$$V_i = \begin{cases} F(BANr, p) & \text{if } v = abstain \\ v & \text{else} \end{cases}$$

   Go to step 1 with V_i as new input value .

Figure 1. *Byzantine Agreement Protocol*

Remark 5 *If a player computes abstain at any step, then he does not send anything in this step. If a player P_i does not send anything to another player P_j, then the latter player regards P_i's vote as abstain. Moreover, if a player cannot do what he is asked (produce a valid signature), then he does not send anything, he simply waits for the next step.*

Theorem 6 *The above protocol achieves Consensus, allowing a negligible probability of error, in constant number of rounds and is 1/2 resilient. If we run the protocol for $4k$ rounds, then the probability of error is $1/2^k$. In addition the bit complexity is BC=$O(kn^2|s|)$, where $|s|$ is the size of a signature share and k a security parameter (number of phases).*

Proof: We decide the round complexity of the protocol by choosing any k we want, however, the largest k is, the smallest the probability of error will be. The bit complexity is also obvious to be BC=$O(n^2|s|)$, but it is also dependent from

$BANr$, because it has to be different every time we run the protocol, yet if we assume that we will not need too many invocations of the protocol we have the latter complexity.

Validity: Assume that all correct players have 1 as their input value. Since the total number of correct players is the same with the corrupted plus at least one, all correct players compute the same value during the second step. Thus no corrupted player will be able to compute a proper vote for 0 during the second step and, hence, all correct players will compute the value 1 in the third step. At last, since no correct player will change his input value in the common-coin step validity will hold no matter how large k is. The same hold if the input value of all correct players was 0.

Consensus: We are going to prove Consensus and see where the probability of error lies examining all possible cases step by step.

Assume that all correct players compute $abstain$ in step 2. Then all correct players will compute $abstain$ in step 3, because the adversary will not have a sufficient number of votes to send to a player and convince him to compute $v \in \{0,1\}$ in the third step. This leads all correct players to have as input value in the next phase the result of the common coin. Then Consensus is satisfied due to validity.

Assume that two different players P_1, P_2 compute b_1 and $b_2 = 1 - b_1$ in second step. Then all correct players will compute $abstain$ during the third step, because each player receives proper messages for both values. Then all players will have as input value in the next phase the result of the common coin and consistency holds because of validity.

At last assume that there are some player, who compute 1 in step 1 and all the other correct players compute $abstain$, then either all players compute $abstain$ in step 3 or some of them (maybe all) compute 1 in step 3. This holds, because the adversary will not have a sufficient number of votes for 0 to make a valid Threshold Signature and send it to an uncorrupted player and make him compute 0 in the third step. This means, that if the result of the common coin is also 1, then all correct players will begin the next phase with input value 1 and consistency holds due to validity.

If all processors were honest, then the protocol achieves Consensus always, because all players would compute the same value during the first step. Only the adversary could make the protocol fail, if he knew the result of the common coin. Knowing the result of the common coin, then the adversary could be able to make some players compute the complementary value of the coin in main-vote step. This would make the correct players not to enter the new phase with the same value. However, if we assume that the adversary has no advantage in predicting the result of the common coin, the probability that the protocol fails is $1/2^k$, where k the pre-decided number of phases.

5. Comparison Table and Conclusions

In table 1 we compare this protocol with others of all cases, information-theoretic, cryptographic, deterministic or random protocols. The comparison is maid in resiliency, Round and Bit Complexity with the most efficient (classical) Byzantine Agreement protocols. The protocols of [Rab83] and [Tou84], which were the most efficient Monte Carlo cryptographic protocols, were not as efficient as the present, because, as we see, they either required a higher Bit Complexity or were less resilient. However, as we saw, it was the technique of Threshold Signatures (of [Sho00]) along with the Common Coin protocol (of [CKS00]), which allowed this protocol to be so efficient.

Table 1.*The complexities of some popular Byzantine Agreement (Consensus)protocols for synchronous networks. By PKI it is meant that there is some consistently shared data before the beginning of the protocol, for example a digital signature scheme or a third trusted party. Note that the protocol with the asterisk (*) demand a stronger trusted dealer, who may be needed more than once as we have already mentioned in the introduction. The protocol with the double asterisk uses a pseudosignature scheme.*

Security	Protocol	Resilience	RC	BC
Information-theoretic	[LSP82]	$t < n/3$	$t+1$	$O(n^t)$
	[BGP89]	$t < n/3$	$3t+1$	$O(tn^2)$
	[BGP92]	$t < n/3$	$t+o(t)$	$O(n^2)$
	[CW92]	$t < n/3$	$t+o(t)$	$O(n^2)$
	[GM98]	$t < n/3$	$t+1$	$O(n^7 t^3)$
Las Vegas	[Bra87]	$(3+\epsilon)t \leq n$	$O(\log n)$	$O(n^5)$
	[FM88]	$t < n/3$	$O(1)$	$O(t^6)$
PKI, deterministic	[DS83]	$t < n$	$t+1$	$O(n^2 t)$
PKI, Monte Carlo	[PW96]**	$t < n/2$	$t+1$	$O(n^3\|\sigma\|)$
	[Rab83]*	$t < n/3$	$O(1)$	$O(n^2\|\sigma\|)$
	[Tou84]*	$t < n/2$	$O(1)$	$O(n^3\|\sigma\|)$
	This protocol	$\mathbf{t < n/2}$	$\mathbf{O(1)}$	$\mathbf{O(n^2\|\sigma\|)}$

6. Acknowledgements

I would like to thank my supervisor Pr. Stathis Zachos and Pr. Aris Pagourtzis for fruitful discussions and helpful comments.

References

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *FOCS*, pages 410–415, 1989.

[BGP92] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus (extended abstract). In *WDAG*, pages 221–237, 1992.

[Bra87] Gabriel Bracha. An o(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *PODC*, pages 123–132, 2000.

[CW92] Brian A. Coan and Jennifer L. Welch. Modular construction of a byzantine agreement protocol with optimal message bit complexity. *Inf. Comput.*, 97(1):61–85, 1992.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *STOC*, pages 148–161, 1988.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $>$ processors in $+ 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[PW96] B. Pfitzmann and M Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. *Technical report RZ 2882 (90830), IBM Research*, 1996.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *FOCS*, pages 403–409, 1983.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sho00] Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.

[Tou84] Sam Toueg. Randomized byzantine agreements. In *PODC*, pages 163–178, 1984.