Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography[1]

Christian Cachin Klaus Kursawe Victor Shoup

IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{cca,kku,sho}@zurich.ibm.com

June 26, 2000

Abstract

Byzantine agreement requires a set of parties in a distributed system to agree on a value even if some parties are corrupted. A new protocol for Byzantine agreement in a completely asynchronous network is presented that makes use of cryptography, specifically of threshold signatures and coin-tossing protocols. These cryptographic protocols have practical and provably secure implementations in the "random oracle" model. In particular, a coin-tossing protocol based on the Diffie-Hellman problem is presented and analyzed.

The resulting asynchronous Byzantine agreement protocol is both practical and theoretically nearly optimal because it tolerates the maximum number of corrupted parties, runs in constant expected time, has message and communication complexity close to the optimum, and uses a trusted dealer only in a setup phase, after which it can process a virtually unlimited number of transactions.

The protocol is formulated as a transaction processing service in a cryptographic security model, which differs from the standard information-theoretic formalization and may be of independent interest.

Keywords: Asynchronous Consensus, Byzantine Faults, Threshold Signatures, Cryptographic Common Coin, Dual-Threshold Schemes.

1 Introduction

The (binary) Byzantine agreement problem is one of the fundamental problems in distributed fault-tolerant computing. In this problem, there are n communicating parties, at most t of which are corrupted. The goal is that all honest (i.e., uncorrupted) parties agree on one of two values that was proposed by an honest party, despite the malicious behavior of the corrupted parties. This problem has been studied under various assumptions regarding the synchrony of the network, the privacy of the communication channels, and the computational power of the corrupted parties.

[1]A preliminary version of this work was presented at the 19th ACM Symposium on Principles of Distributed Computing (PODC), Portland, Oregon, July 2000, and an extended abstract appears in the proceedings.

In this paper, we work exclusively in an *asynchronous* environment with computationally bounded parties; our motivation for this is a secure distributed system connected by the Internet.

Fischer, Lynch, and Paterson (FLP) [23] have shown that no deterministic protocol can guarantee agreement even against benign failures in the asynchronous setting. Rabin [32] and Ben-Or [6] were the first to present protocols that overcome this limitation by using randomization. They assume a *common coin*, a random source observable by all participants but unpredictable for an adversary; this abstraction is used in most subsequent protocols for the asynchronous model.

Our main contributions are an agreement protocol and a common coin protocol that employ modern cryptographic techniques to a far greater extent than has been done previously in the literature. The basic cryptographic primitives used are a non-interactive threshold signature scheme and a novel threshold, random-access coin-tossing scheme. We use dual-threshold variants of both primitives. They can be efficiently implemented and proved secure under standard intractability assumptions in the *random oracle* model; in this model, one treats a cryptographic hash function as if it were a black box containing a random function.

Taken together, we obtain a new protocol for Byzantine agreement that is both practical and theoretically nearly optimal with respect to the known lower bounds because

- it withstands the maximum number of corrupted parties: $t < n/3$;
- it runs in constant expected time;
- the expected number of messages is $O(n^2)$;
- each message is roughly the size of one or two RSA signatures (with the RSA threshold signature scheme of Shoup [38]);
- it uses a trusted dealer only in a setup phase, after which it can process a virtually unlimited number of transactions.

This last point deserves further elaboration. The initial setup phase of our scheme requires a trusted dealer to distribute certain cryptographic keys. Once in place, however, our scheme provides a *transaction processing service* that can handle a virtually *unlimited* number of requests as generated by clients. Moreover, transactions can be processed *concurrently*, i.e., a new instance of the agreement protocol can start as soon as a new transaction request is generated by a client, even if there are extant instances of the protocol for other transactions. This is a non-trivial but important feature for any cryptographic protocol because it rules out so-called interleaving attacks.

1.1 Techniques

Our protocol uses non-interactive threshold signatures and a random-access coin-tossing scheme from cryptography; these have efficient implementations in the random oracle model.

The *random oracle model* was first used in a rather informal way by Fiat and Shamir [22]; it was first formalized and used in other contexts by Bellare and Rogaway [4] and has since been used to analyze a number of practical cryptographic protocols. Of course, it would be better not to rely on random oracles, as they are essentially a heuristic device; nevertheless, random oracles are a useful tool—they allow us to design truly practical protocols that admit a security analysis, which yields very strong evidence for their security. As far as we know, our work is the first of its kind to apply the random oracle model to the Byzantine agreement problem.

The notion of a *threshold signature scheme* was introduced by Desmedt, Frankel and others [17, 18, 8, 16] and has been widely studied since then (T. Rabin [33] provides new results and a survey of recent literature). It is a protocol for n parties tolerating up to t corruptions, where each party holds a share of the signing key and k cooperating parties together can generate a signature. In a *non-interactive* threshold signature scheme, each party outputs a signature share upon request and there is an algorithm to combine k valid signature shares to constitute a valid signature. Such non-interactive combination is used in our agreement protocol: a party can justify its vote for a particular value by a single threshold signature generated from k signature shares. This saves a factor n in terms of bit complexity.

One of the technical contributions of this paper is the notion of a *dual-threshold signature scheme*, meaning that k is allowed to be higher than $t+1$. This is in contrast to all previous work on threshold signatures in the literature where $k = t+1$. A companion paper [38] presents a practical dual-threshold signature scheme that is secure in the random oracle model under standard intractability assumptions. The signatures created by this scheme are ordinary RSA signatures. Moreover, the scheme is completely non-interactive, an individual share of a signature is not much greater than an ordinary RSA signature, and even for $k = t+1$, it is the first rigorously analyzed non-interactive threshold signature scheme with small shares.

Coin-tossing schemes are used in one form or another in essentially all solutions to the asynchronous Byzantine agreement problem. Many schemes, following Rabin's pioneering work [32], assume that coins are predistributed (and possibly signed) by a dealer using secret-sharing [35]. This approach has two problems: first, the coins will eventually be exhausted; second, parties must somehow associate coins with transactions, which itself represents an agreement problem. Because of these problems, protocols that rely on a "Rabin dealer" are not really suitable for use as a transaction processing service as described here. The same applies to the coin-tossing scheme of Beaver and So [2], which essentially gives parties sequential access to a bounded number of coins. A "Rabin dealer" has been used in other contexts as well, e.g., for threshold decryption [11]. Our protocol also requires a dealer for the initial setup, but yields an arbitrary polynomial number of coins afterwards.

The beautiful work of Canetti and T. Rabin [12] presents a coin-tossing scheme that allows common coins to be generated entirely "from scratch," building on the work of Feldman and Micali for the synchronous model [21]. Unfortunately, this scheme, while polynomial time, is completely impractical.

Our approach to coin-tossing is to use a random-access coin-tossing scheme—essentially a distributed function mapping the "name" of a coin to its value. Such coin-tossing schemes have been studied before [29, 30]. We also define the notion of a *dual-threshold coin-tossing scheme*, which is convenient and does lead to lower communication complexity, but is not absolutely necessarily. One could easily implement such a coin from the non-interactive threshold signature scheme of Shoup [38]; however, we present a dual-threshold coin-tossing scheme that is based on the Diffie-Hellman problem, the analysis of which may be interesting in its own right. This scheme is essentially the same as the one of Naor et al. [30], but our analysis is more refined: first, for the single-parameter setting, we need a weaker intractability assumption, and second, we provide an analysis of the scheme in the dual-threshold setting, which is not considered by Naor et al.

We stress that such dual-parameter threshold schemes provide stronger security guarantees than single-parameter threshold schemes, and they are in fact more challenging to construct and to analyze. Our notion of a dual-threshold scheme should not be confused with a weaker notion that sometimes appears in the literature (e.g., [29]). For this weaker notion, there is a parameter $l > t$ such that the reconstruction algorithm requires l shares, but the security guarantee for a given signature/coin is lost if just a *single* honest party reveals a share. In our notion, no security is lost unless $k - t$ honest parties reveal their shares.

1.2 Related Work

The problem of asynchronous Byzantine agreement has a long history—see the survey of the early Byzantine era by Chor and Dwork [15] and the more recent account by Berman and Garay [7]. A fundamental result in this area is the impossibility result of Fischer, Lynch, and Paterson [23] that rules out the existence of a deterministic protocol. The protocols of Rabin [32] and Ben-Or [6] are the first probabilistic protocols to overcome this limitation. Bracha's protocol improves the resilience to the maximum $t < n/3$ [9].

We shall compare our protocol to others in the literature on several criteria. For these purposes, it is sufficient to consider the protocols of Bracha [9], Toueg [40], Berman and Garay (BG) [7], and Canetti and Rabin (CR) [12] (see [10] for details). The protocols of Toueg [40] and BG [7] can be seen as descendants of Rabin's pioneering work [32], whereas Bracha [9] and CR [12] can be viewed as descendants of Ben-Or's initial randomized algorithm [6]; CR [12] also builds on ideas of Feldman and Micali [21] and Bracha [9].

These protocols vary in a number of aspects:

Resilience: how many parties may be corrupted. The theoretical maximum is $t < n/3$, which is attained by our protocol, as well as the protocols of Toueg [40], Bracha [9], and CR [12]. The BG protocol [7] handles $t < n/5$.

Time Complexity: the (expected) number of basic steps before a decision is reached. Our protocol, like those of Toueg [40], Bracha [12], and BG [7], has complexity $O(1)$. The protocol of Bracha [9] takes exponential time when $t = \Theta(n)$ and expected constant time if $t = O(\sqrt{n})$.

Message Complexity: the (expected) number of messages sent during the protocol. Our protocol has a message complexity of $O(n^2)$. All the other protocols in the literature with an $O(n^2)$ bound, such as BG [7], do not achieve optimal resilience; the protocol of Toueg [40] has a message complexity of $O(n^3)$, and the CR protocol [12] has a message complexity that is completely impractical (although polynomial in n), which renders it to be of theoretical interest only.

Bit Complexity: the (expected) total bit-length of messages during the protocol. Our protocol has a bit complexity of $O(n^2 l)$, where l is the length of an RSA signature; the protocol of BG [7] has a bit complexity of $O(n^2 l')$, where l' is the length of a message authentication code (typically significantly less than the size of an RSA signature); in practice, the difference between l and l' is probably irrelevant, as in both cases, all messages easily fit into a single IP packet.

Computational Complexity: the (expected) amount of computation that must be done locally by each party. Most papers on this subject do not make very careful estimates of computational complexity; however, a useful distinction can be made between protocols, like ours and Toueg's [40], that use (typically expensive) public-key cryptography, and those that do not [9], [7], [12].

Dealer: the degree to which a single trusted "dealer" is involved. Possible models are

no dealer: No dealer is needed [9], [12].

system setup dealer: A dealer is needed to set up the initial states of parties, but after this, an effectively unlimited number of transactions may be processed. Our protocol is of this type; depending on how secure channels are implemented, many other protocols in the literature may implicitly fall in this category as well.

Rabin dealer: Each transaction requires data that was pre-distributed by the dealer among the parties [7] [40]. All of this data must be stored by each processor and this pre-distributed data will be exhausted eventually. Moreover, the parties must agree on which data to use for a given transaction. These drawbacks render such protocols unsuitable for many applications that require a transaction processing service.

Computation Model: the computational power of the adversary. It can be

bounded: The adversary is constrained to perform only polynomial-time computations and one must make specific assumptions about the intractability of certain problems. This is our model, as well as the (implicit) model of Toueg [40].

unbounded: The adversary is computationally unlimited. In this case, one must explicitly assume that channels are secure (authenticated, and perhaps private), since they cannot be secured by cryptography [9], [7], [12].

Corruption Model: how the adversary decides to corrupt parties. This can be

static: The adversary's choice of who to corrupt is independent of the network traffic. This is our model.

adaptive: The adversary chooses who to corrupt adaptively, based upon the network traffic so far and the internal states of previously corrupted parties. This is the model of CR [12], and is also implicit in the others [9], [40], [7].

Many authors like to classify agreement protocols based on whether they use digital signatures or not. We do not see this distinction as a fundamental one, although the use of signatures definitely impacts the computation model, and can also affect the computational and bit complexity.

There is also a line of research which attempts to avoid the use of probabilistic protocols, despite FLP [23]. For example, Reiter [34], adapts the approach of "failure detectors" [13, 41] used in the asynchronous crash-failure model to the asynchronous Byzantine setting. Reiter presents a protocol for *atomic broadcast*, from which a Byzantine agreement protocol can be constructed (see [19]). However, as Reiter's protocol is deterministic, the FLP result implies that it can not solve the Byzantine agreement problem. In fact, Reiter's protocol ensures correctness only as long as the network is suitably well behaved—it is easily defeated by an adversary that completely controls network scheduling. Indeed, it has been recognized that extending the modular failure detector approach to the Byzantine model is difficult (e.g., [20]).

Reiter's work [34], and related work, seems to be motivated by the fact that probabilistic agreement protocols have a reputation for being impractical. However, it is not at all clear if this reputation is well justified—we know of no empirical, comparative studies in the literature. Much of the confusion arises because almost all of the work on probabilistic protocols has been done by researchers who have been more interested in theoretically attractive, rather than practical results. Their ground rules might not even consider our use of random oracles in the protocol analysis as legal.

From an efficiency point of view, the strongest criticism of our new protocol is its use of somewhat expensive public-key cryptography. However, even this can be avoided using an "optimistic" approach that uses public-key cryptography only as a "fall back" mechanism when some parties crash or misbehave, or the network is temporarily slower than expected. Such an approach, developed in a companion paper [26], seems an attractive alternative to failure detectors.

1.3 Motivation

Malicious attacks are increasingly common on the Internet. Despite the growing reliance of industry and government on electronic forms of conducting business, system failures resulting from attacks or software errors are reported almost daily. Fault-tolerant distributed systems have long been recognized as a possible solution, but only few of the many theoretical solutions are applicable to the Internet setting. For one thing, synchronization is difficult to guarantee on the Internet and one must therefore work in an asynchronous model. Another difficulty is that one faces potentially malicious adversaries, who seem to get some benefit from disrupting or, even more so, from subverting a service. This motivates the choice of the Byzantine failure model as the only one that can guarantee service integrity under clearly defined assumptions that include malicious attacks.

Our initial motivation for studying this problem was to design a distributed trusted third-party service to be used in the fair exchange and contract signing protocols presented by Asokan, Shoup, and Waidner [1]. In that setting, the trusted third party must make a decision to either "abort" or "resolve" a transaction at the request of one of the parties involved in the exchange. If one distributes the service so as to weaken the necessary trust assumption, a Byzantine agreement problem has to be solved. As attacks may very well involve the administrators of the computing systems implementing the distributed service, the service should consist of independently administered and geographically distributed computing systems.

The trusted third-party service is a prime application for the method of increasing the security guarantees of a service by fault-tolerant computation; we believe that this will become an important paradigm for secure Internet applications.

1.4 Organization

In §2 we introduce our asynchronous system model using cryptography. §3 contains the definition of Byzantine agreement and §4 introduces the cryptographic primitives of threshold signatures and coin-tossing protocols. The agreement protocol based on these primitives is presented in §5 and our coin-tossing protocol is given in §6.

2 Basic System Model

In this section, we describe our basic system model for an arbitrary multi-party protocol where a number of parties communicate over an insecure, asynchronous network, and where an adversary may corrupt some of the parties. Our point of view is *computational*: all parties and the adversary are constrained to perform only feasible computations. This differs substantially from the traditional secure channels model in distributed computing, but is necessary and also appropriate for the cryptographic setting (cf. [5, 3, 37]). Although authentication and digital signatures have been used before in agreement protocols, there seem to be no adequate cryptographic formal models [28, p. 115].

There are n parties, $P_1, \ldots, P_n$, an adversary that is allowed to corrupt up to t of them, and a trusted "dealer."

We adopt the *static* corruption model, wherein the adversary must decide whom to corrupt at the very outset of the execution of the system. Let f, with $0 \le f \le t$, denote the number of parties the adversary actually corrupts. These corrupted parties are simply absorbed into the adversary: we do not regard them as system components.

Alternatively, one could adopt the *adaptive* corruption model, wherein the adversary can adaptively choose whom to corrupt as the attack is ongoing, based on information it has accumulated so far. We do not adopt this model, mainly because we would no longer know how to obtain the practical, provably secure implementations of the necessary cryptographic primitives. Moreover, the static corruption model is not too unrealistic; in practice, the choice of whom to corrupt is usually based on factors totally independent of the network traffic (e.g., which system administrator is not careful, or can perhaps be bribed or blackmailed).

There is an initial setup phase, in which the trusted dealer generates the initial state for all n parties. The adversary obtains the initial state of the corrupted parties, but obtains no information about the initial state given to the honest parties.

Our network is insecure and asynchronous, i.e., the adversary has complete control of the network: he may schedule the delivery of messages as he wishes, and may modify or insert messages as he wishes. As such, the network is merely absorbed into the adversary in our formal model. The honest parties are completely passive: they simply *react* to requests made by the adversary and maintain their internal state between requests. More precisely, after the initial setup phase, the adversary performs a number of *basic steps*. One basic step works as follows: the adversary delivers a message to an honest party P_i; then P_i updates its internal state, and computes a set of response messages; these messages are then given to the adversary. These response messages perhaps indicate to whom these messages should be sent, and the adversary may choose to deliver these messages faithfully at some time. In general, the adversary chooses to deliver any messages it wants, or no messages at all; we may sometimes impose additional restrictions on the adversary's behavior, however.

Of course, the computations made by the honest parties, the adversary, and the dealer should all be representable as probabilistic, polynomial-time computations. To be completely formal, we would have to introduce a security parameter, and all the computations would be bounded by a polynomial in this security parameter. In particular, the parameter n and the number of basic steps performed by the adversary are polynomially bounded in the security parameter.

The dealing algorithm and the algorithm executed locally by each P_i to compute its new state and response messages are specific to the particular protocol. The dealing algorithm is given the security parameter, as well as n and t as input. Note that the adversary chooses n and t, but a specific protocol might impose its own restrictions (e.g., $t < n/3$). We can assume that the dealer includes these values, as well as the index i, in the initial state of P_i.

3 Definition of Byzantine Agreement

We now define the operation and requirements of a Byzantine agreement protocol, in the context of our basic system model described in the previous section. There are n parties, $P_1, \dots, P_n$, and the adversary may corrupt some number f of them, where $f \le t$.

As mentioned in the introduction, we want an agreement protocol that can be used to implement a *transaction processing service*. To this end, we assume that each decision to be made is associated with a unique transaction identifier *TID*. The value *TID* is an arbitrary bit string whose structure and meaning are determined by a particular application. In our formal model, it is simply chosen by the adversary.

The adversary may deliver a message to P_i of the form

$$(\mathit{TID}, \texttt{activate}, \mathit{initial\ value}),$$

where *initial value* is in $\{0, 1\}$. When the adversary has delivered such a message, we say that P_i is *activated* on *TID* with the given *initial value*. After activating P_i on *TID*, the adversary may then deliver messages to P_i of the form

$$(\mathit{TID}, j, i, \dots),$$

where $1 \leq j \leq n$ denotes the index of the sender.

Upon receiving a message involving *TID*, P_i updates its internal state, and generates a set (possibly empty) of response messages. Each of this messages is either of the form

$$(\mathit{TID}, i, j, \dots),$$

where $1 \leq j \leq n$ denotes the index of the recipient, or

$$(\mathit{TID}, \texttt{decide}, \mathit{final\ value}),$$

where *final value* $\in \{0, 1\}$. In the latter case, we say that P_i *decides final value* for *TID*. We require that P_i makes a decision for a given *TID* at most once. However, the adversary may continue to deliver messages involving *TID* after P_i has made a decision for *TID*.

For simplicity, we shall assume that messages are *authenticated*, which means that we restrict the adversary's behavior as follows: if P_i and P_j are honest, and the adversary delivers a message M of the form $(\mathit{TID}, i, j, \dots)$ to P_j, then the message M must have been generated by P_i at some prior point in time. It is reasonable to build authentication into our model because it can be implemented very cheaply using standard symmetric-key cryptographic techniques.

The three basic properties that an agreement protocol must satisfy are *agreement*, *termination*, and *validity*.

Agreement. Any two honest parties that decide a value for a particular *TID* must decide the same value. More precisely, it is computationally *infeasible* for an adversary to make two honest parties decide on different values.

Termination. The traditional approach in the distributed computing literature is to assume that all messages between honest parties are "eventually" delivered, and then to define the termination condition to be that all honest parties "eventually" decide (with probability 1). In formalizing these definitions, one considers infinite runs of a protocol; however, in the computationally bounded setting, this simply does not work.

We present here a workable definition in our setting that captures the intuition that *to the extent* the adversary delivers messages among honest parties, the honest parties *quickly* decide. Although the intuition is fairly clear, one has to be careful with the details. For us, *termination* consists of two conditions: *deadlock freeness* and *fast convergence*.

Deadlock freeness. It is infeasible for the adversary to create a situation where for some *TID* there are some honest parties who are not decided, yet all honest parties have been activated on *TID*, and all messages relating to this *TID* generated by honest parties have been delivered.

Fast Convergence. For $s = 1, 2, \ldots$, let TID_s denote the sth transaction identifier introduced by the adversary, and define X_s to be the total number of messages *generated* by all honest parties that relate to TID_s. Then there exist fixed polynomials B and C in n and in the security parameter such that for all $s \geq 1$ and $m \geq 1$,

$$\Pr[X_s \geq mB + C] \leq 2^{-m} + \epsilon,$$

where ϵ is a function that is negligible in the security parameter (i.e., it vanishes faster than any polynomial in the security parameter). Note that while ϵ may depend on the adversary, the polynomials B and C depend only on the agreement protocol, and are independent of the adversary.

The *deadlock freeness* property rules out trivial protocols that never decide and never generate any messages to be delivered. The *fast convergence* property ensures timely convergence, provided the adversary delivers messages; also, the fact that B and C are independent of the adversary rules out trivial protocols that never decide but always generate "make work" messages to be delivered.

Our definition of *termination* implies that an adversary *could* quickly make all honest parties make a decision on a given *TID* (with probability exponentially close to 1) by delivering a (fixed) polynomially bounded number of messages; however, we do not *force* the adversary to do so—see [10] for a definition more along these lines.

Validity. If all honest parties are activated on a given *TID* with the same initial value, then any honest party that decides must decide this value.

This is the usual definition of validity in the literature. A weaker notion of validity may sometimes be more appropriate for particular applications. For instance, initial values may come with validating data (e.g., a digital signature) that establishes the "validity" of a value in a particular context. One could then simply require that an honest party may only decide on a value for which it has the accompanying validating data—even if all honest parties start with 0, they may still decide on 1 if they obtain the corresponding validating data for 1 during the agreement protocol.

4 Cryptographic Primitives

4.1 Digital Signatures

A digital signature scheme [25] consists of a *key generation* algorithm, a *signing* algorithm, and a *verification* algorithm. The key generation algorithm takes as input a security parameter, and outputs a public key/private key pair (PK, SK). The signing algorithm takes as input SK and a message M, and produces a signature σ. The verification algorithm takes PK, a message M, and a putative signature σ, and outputs either `accept` or `reject`. A signature is considered *valid* if and only if the verification algorithm accepts. All signatures produced by the signing algorithm must be valid.

The basic security property is *unforgeability.* The attack scenario is as follows. An adversary is given the public key, and then requests the signatures on a number of messages, where the messages themselves may depend on previously obtained signatures. If at the end of the attack, the adversary can output a message M and a valid signature σ on M, such that M was not one of the messages whose signature he requested, then the adversary has successfully *forged* a signature. Security means that it is computationally infeasible for an adversary to forge a signature.

4.2 Threshold Signatures

In this section, we define the notion of an (n, k, t) dual-threshold signature scheme. The basic idea is that there are n parties, up to t of which may be corrupted. The parties hold *shares* of the secret key of a signature scheme, and may generate *shares of signatures* on individual messages—k signature shares are both necessary and sufficient to construct a signature. The only requirement on k is that $t < k \leq n - t$. As mentioned in the introduction, previous investigations into threshold signatures have only considered the case $k = t + 1$. Also, we shall require that the generation and verification of signature shares is completely non-interactive—this is essential in the application of asynchronous Byzantine agreement.

A threshold signature scheme is a multi-party protocol, and we shall work in our basic system model for such protocols (see §2).

The Action. The dealer generates a public key PK along with secret key shares $SK_1, \ldots, SK_n$, a global verification key VK, and local verification keys $VK_1, \ldots, VK_n$. The initial state information for party P_i consists of the secret key SK_i along with the public key and *all* the verification keys.

After the dealing phase, the adversary submits signing requests to the honest parties for messages of his choice. Upon such a request, party P_i computes a *signature share* for the given message using SK_i.

Combining Signature Shares. The threshold signature scheme also specifies three algorithms: a *signature verification* algorithm, a *share verification* algorithm, and a *share combining* algorithm.

- The *signature verification* algorithm takes as input a message and a signature (generated by the share-combining algorithm), along with the public key, and determines if the signature is valid.

- The *share verification* algorithm takes as input a message, a signature share on that message from a party P_i, along with PK, VK, and VK_i, and determines if the signature share is valid.

- The *share combining* algorithm takes as input a message and k valid signature shares on the message, along with the public key and (perhaps) the verification keys, and (hopefully) outputs a valid signature on the message.

Security Requirements. The two basic security requirements are *robustness* and *non-forgeability.*

Robustness. If it computationally infeasible for an adversary to produce k valid signature shares such that the output of the share combining algorithm is not a valid signature.

Non-forgeability. It is computationally infeasible for the adversary to output a valid signature on a message that was submitted as a signing request to less than $k - t$ honest parties. Note that if the adversary actually corrupts $f < t$ parties, the relevant threshold is still $k - t$ and not $k - f$.

Implementation. Note that our definition of a threshold signature scheme admits the trivial implementation of just using a set of k ordinary signatures. For relatively small values of n, this may very well be a perfectly adequate implementation. (Such a scheme cannot be used to implement the coin-tossing scheme, however.)

The scheme of Shoup [38] is well suited to our purposes and is much more efficient than the above trivial implementation when n gets large. Each signature share is essentially the size of an RSA signature, and shares can be quite efficiently combined to obtain a completely standard RSA signature. The signature shares come with "proofs of correctness." These correctness proofs are not much bigger than RSA signatures; however, in an efficient implementation, one would most likely omit these proofs (and their verification), and only provide them if they are explicitly requested, presumably by a party whose share combination algorithm has failed to produce a correct signature.

4.3 Threshold Coin-Tossing Scheme

In this section, we define the notion of an (n, k, t) dual-threshold coin-tossing scheme. The basic idea is that there are n parties, up to t of which may be corrupted. The parties hold *shares* of an unpredictable function F mapping the name C (which is an arbitrary bit string) of a coin to its value $F(C) \in \{0, 1\}$. The parties may generate *shares of a coin*—k coin shares are both necessary and sufficient to construct the value of the particular coin. The only requirement on k is that $t < k \leq n - t$, analogous to threshold signatures. The generation and verification of coin shares are completely non-interactive; we work in the basic system model of §2.

The Action. The dealer generates secret key shares $SK_1, \ldots, SK_n$, and verification keys $VK, VK_1, \ldots, VK_n$. The initial state information for party P_i consists of the secret key SK_i along with all the verification keys. The secret keys implicitly define a function F mapping names to $\{0, 1\}$.

After the dealing phase, the adversary submits *reveal requests* to the honest parties for coins of his choice. Upon such a request, party P_i outputs a *coin share* for the given coin, which it computes using SK_i.

Combining Coin Shares. The coin-tossing scheme also specifies two algorithms: a *share verification* algorithm, and a *share combining* algorithm.

- The *share verification* algorithm takes as input the name of a coin, a share on this coin from a party P_i, along with VK and VK_i, and determines if the coin share is valid.
- The *share combining* algorithm takes as input a the name C of a coin and k valid shares of C, along with (perhaps) the verification keys, and (hopefully) outputs $F(C)$.

Security Requirements. The two basic security requirements are *robustness* and *unpredictability.*

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

Unpredictability. An adversary's *advantage* in the following game is negligible. The adversary interacts with the honest parties as above, and at the end of this interaction, he outputs a name C that was submitted as a reveal request to fewer than $k - t$ honest parties, and a bit $b \in \{0, 1\}$. The adversary's advantage in this game is defined to be the distance from $1/2$ of the probability that $F(C) = b$. Note that if the adversary actually corrupts $f < t$ parties, the relevant threshold is still $k - t$ and not $k - f$.

Unpredictability for Sequences of Coins. The unpredictability property above implies the following more general unpredictability property that we actually need in order to analyze agreement protocols.

Consider an adversary A that interacts with the honest parties as above, but as it interacts, it makes a sequence of predictions, predicting $b_i \in \{0,1\}$ as the value of coin C_i for $i = 1, \ldots, q$ for some q. A's predictions are interleaved with reveal requests in an arbitrary way, subject only to the restriction that at the point in time that A predicts the value of coin C_i, it has made fewer than $k - t$ reveal requests for C_i. After it predicts C_i, it may make as many reveal requests for C_i as it wishes. For $1 \le i \le q$, let $e_i = F(C_i) \oplus b_i$. This defines the *error vector* $(e_1, \ldots, e_q)$.

The unpredictability property above implies that the error vector is *computationally indistinguishable* from a random bit-vector of length q. This means that there is no effective statistical test that distinguishes the error vector from a random vector—the important point is that we are considering statistical tests that receive *only* the test vector as input, and no additional information about A's interaction in the above game.

A proof of this can be adapted easily from the work of Beaver and So [2], although their setting is slightly different. The idea of the proof runs as follows. By the universality of the next-bit test [42], if the error vector were distinguishable from a random vector, then there would be an algorithm D that on input j, chosen randomly from $\{1, \ldots, q\}$, along with $e_1, \ldots, e_{j-1}$, outputs a value that correctly predicts e_j with probability significantly better than 1/2. Given this D and A, we construct a new adversary A' that predicts a single coin, contradicting the unpredictability assumption. A' runs as follows. First, it chooses $j \in \{1, \ldots, q\}$ at random. Next, it runs A as a subroutine. Just after A predicts coin C_i for $1 \le i < j$, A' immediately makes a sufficient number of reveal requests to obtain $F(C_i)$, and hence e_i. A' stops A just after A makes its prediction b_j for the value of $F(C_j)$, and then A' computes

$$\hat{b}_j = D(j; e_1, \ldots, e_{j-1}) \oplus b_j$$

as its prediction for $F(C_j)$ and halts. It is easy to see that $\hat{b}_j$ is correct with probability significantly better than 1/2.

Given the pseudo-random quality of the error vector, one can now easily derive a number of simple statistical properties. The only we will need is this: for any $1 \le m \le q$, the probability that A correctly predicts the first m coins is bounded by $2^{-m} + \epsilon$, where ϵ is a negligible function in the security parameter.

Implementation. Note that an implementation of a coin-tossing scheme can be obtained from any non-interactive threshold signature scheme with the property that there is only one valid signature per message, such as the RSA-based scheme mentioned earlier [38]. Then a cryptographic hash of the signature can be used as the value of the coin. It is straightforward to see that in the random oracle model, this yields a secure coin-tossing scheme. It also allows an implementation to "optimistically" skip the verification tests unless necessary.

In §6 we present also a direct implementation of a coin-tossing scheme based on the Diffie-Hellman problem.

5 Asynchronous Byzantine Agreement

5.1 Protocol ABBA

We now present our protocol ABBA, which stands for Asynchronous Binary Byzantine Agreement. As usual there are n parties $P_1, \ldots, P_n$, up to t of which may be corrupted by the adversary. We denote by f the actual number of parties corrupted.

The protocol uses an $(n, n-t, t)$ threshold signature scheme $\mathcal{S}$ and an $(n, t+1, t)$ threshold signature scheme $\mathcal{S}_0$ (see §4.2), as well as an $(n, n-t, t)$ threshold coin-tossing scheme (see §4.3). Let $F(C)$ denote the value of coin with name C.

Overview. For a given transaction identifier *TID*, each party P_i has an initial value $V_i \in \{0, 1\}$, and the protocol proceeds in rounds $r = 1, 2, \ldots$ The first round starts with a special pre-processing step:

0. Each party sends its *initial value* to all other parties signed with an $\mathcal{S}_0$-signature share. On receiving $2t+1$ such votes, each party combines the signature shares of the value with the simple majority (i.e., at least $t+1$ votes) to a threshold signature of $\mathcal{S}_0$. This value will be the value used in the first pre-vote. (This step is not necessary if the input values are accompanied by validating data.)

After that each round contains four basic steps:

1. Each party casts a *pre-vote* for a value $b \in \{0, 1\}$. These pre-votes must be *justified* by an appropriate $\mathcal{S}$-threshold signature, and must be accompanied by a valid $\mathcal{S}$-signature share on an appropriate message.

2. After collecting $n-t$ valid pre-votes, each party casts a *main-vote* $v \in \{0, 1, \texttt{abstain}\}$. As with pre-votes, these main-votes must be justified by an appropriate $\mathcal{S}$-threshold signature, and must be accompanied by a valid $\mathcal{S}$-signature share on an appropriate message.

3. After collecting $n-t$ valid main-votes, each party examines these votes. If all votes are for a value $b \in \{0, 1\}$, then the party *decides* b for *TID*, but continues to participate in the protocol for one more round. Otherwise, the party proceeds.

4. The value of coin (TID, r) is revealed, which may be used in the next round.

We now proceed with the details of the protocol given in Figure 1. We first introduce some conventions.

Recall that a message from P_i to P_j has the form $(\mathit{TID}, i, j, \mathit{payload})$, so that in specifying a message, we will only specify the *payload* if necessary; the values of *TID*, i, and j are implied from the context.

The pre-vote and main-vote messages have to contain a proper *justification*, which consists of threshold signatures on collected votes as follows.

Pre-Vote Justification. In round $r = 1$, party P_i's pre-vote is the majority of the pre-processing votes from step 0. There must be at least $t+1$ votes for the same value $b \in \{0, 1\}$ (although this b might not be unique if $n > 3t+1$). For the justification, a party selects $t+1$ such votes, and combines the accompanying $\mathcal{S}_0$-signature shares to obtain an $\mathcal{S}_0$-threshold signature on the message

$$(\mathit{TID}, \texttt{pre-process}, b).$$

In rounds $r > 1$, a pre-vote for b may be justified in two ways:

- either with an $\mathcal{S}$-threshold signature on the message

$$(TID, \texttt{pre-vote}, r-1, b);$$

  we call this a *hard* pre-vote for b;

- or with an $\mathcal{S}$-threshold signature on the message

$$(TID, \texttt{main-vote}, r-1, \texttt{abstain})$$

  for the pre-vote $b = F(TID, r-1)$; we call this a *soft* pre-vote for b.

Intuitively, a hard pre-vote expresses P_i's preference for b based on evidence for preference b in round $r-1$, whereas a soft pre-vote is just a vote for the value of the coin, based evidence of conflicting votes in round $r-1$. The threshold signatures are obtained from the computations in previous rounds (see below). We assume that the justification indicates whether the pre-vote is hard or soft.

Main-Vote Justification. A main-vote v in round r is one of the values $\{0, 1, \texttt{abstain}\}$ and, like pre-votes, accompanied by a *justification* as follows:

- If among the $n-t$ justified round-r pre-votes collected by P_i there is a pre-vote for 0 and a pre-vote for 1, then P_i's main-vote v for round r is `abstain`. The justification for this main-vote consists of the justifications for the two conflicting pre-votes.

- Otherwise, P_i has collected $n-t$ justified pre-votes for some $b \in \{0, 1\}$ in round r, and since each of these comes with a valid $\mathcal{S}$-signature share on the message

$$(TID, \texttt{pre-vote}, r, b),$$

  party P_i can combine these shares to obtain a valid $\mathcal{S}$-threshold signature on this message. Party P_i's main-vote v in this case is b, and its justification is this threshold signature.

The protocol is shown in Figure 1.

5.2 Analysis

Theorem 1 *Assuming a secure threshold signature scheme, a secure threshold coin-tossing scheme, and a secure message authentication code, protocol ABBA solves asynchronous Byzantine agreement for $n > 3t$.*

The rest of this section outlines a proof of this theorem. We have to show *validity*, *agreement*, and *termination*.

It is straightforward to check that protocol ABBA satisfies the *validity* condition.

We prove *agreement* and *termination* assuming the adversary corrupts exactly $f = t$ parties; we then discuss the modifications necesarry for the case that $f < t$.

Fix a given TID and consider the pre-votes cast by honest parties in round $r \geq 1$. Because $n > 3t$, there will be at most one value $b \in \{0, 1\}$ that garners at least $n - 2t$ such pre-votes, and we define ρ_r to be this value (if it exists), and otherwise we say that ρ_r is undefined. We say that ρ_r is defined at the point in the game at which time sufficient pre-votes are cast.

Protocol ABBA for party P_i with initial value V_i

0. PRE-PROCESSING. Generate an $\mathcal{S}_0$-signature share on the message

$$(\mathit{TID}, \texttt{pre-process}, V_i).$$

and send a message of the form

$$(\texttt{pre-process}, V_i, \textit{signature share})$$

to all parties.

Collect $2t + 1$ proper pre-processing messages.

Repeat the following steps 1–4 for rounds $r = 1, 2, \ldots$.

1. PRE-VOTE. If $r = 1$, let b be the simple majority of the received pre-processing votes. Otherwise, if $r > 1$, select $n - t$ properly justified main-votes from round $r - 1$ and let

$$b = \begin{cases} 0 & \text{if there is a main-vote for 0,} \\ 1 & \text{if there is a main-vote for 1,} \\ F(\mathit{TID}, r-1) & \text{if all main-votes are } \texttt{abstain}. \end{cases}$$

Produce an $\mathcal{S}$-signature share on the message

$$(\mathit{TID}, \texttt{pre-vote}, r, b).$$

Produce the corresponding justification (see text) and send to all parties a message of the form

$$(\texttt{pre-vote}, r, b, \textit{justification}, \textit{signature share}).$$

2. MAIN-VOTE. Collect $n - t$ properly justified round-r pre-vote messages. Consider these pre-votes and let

$$v = \begin{cases} 0 & \text{if there are } n-t \text{ pre-votes for 0,} \\ 1 & \text{if there are } n-t \text{ pre-votes for 1,} \\ \texttt{abstain} & \text{if there are pre-votes for 0 and 1.} \end{cases}$$

Produce an $\mathcal{S}$-signature share on the message

$$(\mathit{TID}, \texttt{main-vote}, r, v).$$

Produce the corresponding justification (see text) and send to all parties a message of the form

$$(\texttt{main-vote}, r, v, \textit{justification}, \textit{signature share}).$$

3. CHECK FOR DECISION. Collect $n - t$ properly justified main-votes of round r. If these are all main-votes for $b \in \{0, 1\}$, then *decide* the value b for *TID*, and continue for one more round (up to step 2). Otherwise, simply proceed.

4. COMMON COIN. Generate a *coin share* of the coin (TID, r), and send to all parties a message of the form

$$(\texttt{coin}, r, \textit{coin share}).$$

Collect $n - t$ shares of the coin (TID, r), and combine these shares to get the value $F(\mathit{TID}, r) \in \{0, 1\}$.

Figure 1: Asynchronous Binary Byzantine Agreement

Lemma 2 *For $r \geq 1$, the following holds (with all but negligible probability):*

(a) if an honest party casts or accepts a main-vote of $b \in \{0, 1\}$ in round r, then ρ_r is defined and $\rho_r = b$;

(b) if an honest party casts or accepts a hard pre-vote for $b \in \{0, 1\}$ in round $r + 1$, then ρ_r is defined and $\rho_r = b$;

(c) if an honest party casts or accepts a main-vote of `abstain` *in round $r + 1$, then ρ_r is defined and $\rho_r = 1 - F(TID, r)$;*

(d) if r is the first round in which any honest party decides, then all honest parties that eventually decide, decide the same value in either round r or $r + 1$.

Proof. To prove (a), suppose an honest party accepts a main-vote of $b \in \{0, 1\}$ in round r. To be justified, this main-vote must be accompanied by a valid threshold signature on the message

$$(\texttt{pre-vote}, \mathit{TID}, r, b).$$

By the non-forgeability property of the signature scheme, this implies that at least $(n-t)-t = n - 2t$ honest parties cast pre-votes for b. Thus, ρ_r has been defined and is equal to b. That proves (a).

Part (b) now simply follows from the fact that a hard pre-vote for $b \in \{0, 1\}$ in round $r + 1$ is justified by the same threshold signature as the main-vote from round r in part (a).

Now for part (c). A main vote of `abstain` in round $r + 1$ must be accompanied by a justification for a pre-vote of 0 in round $r+1$ and a justification for pre-vote of 1 in round $r+1$. These pre-votes cannot both be soft pre-votes, and so one of these two pre-votes must be hard. It follows from (b) that this hard pre-vote must be for ρ_r, and hence the other pre-vote must be a soft pre-vote for $1 - \rho_r$, and hence $F(\mathit{TID}, r) = 1 - \rho_r$. Part (c) now follows.

Now for part (d). Suppose some party P_i decides $b \in \{0, 1\}$ in some round r, and no party has decided in a previous round. Then in this round, P_i accepted $n - t$ main-votes for b. By part (a), we must have $b = \rho_r$. So any other honest party who decides in round r must also decide ρ_r.

Of the $n - t$ main-votes for b that P_i accepted, at least $n - 2t$ came from honest parties who main-voted b, and since $n > 3t$, fewer than $(n - t) - t = n - 2t$ signature shares on the message

$$(\texttt{main-vote}, r, \texttt{abstain})$$

have been or ever will be generated by honest parties. This in turn implies that a soft pre-vote in round $r + 1$ cannot be justified. Thus, the only justifiable pre-votes in round $r + 1$ are hard pre-votes, and by part (b), these must be hard pre-votes of b. Finally, this implies that the only justifiable main-votes in round $r + 1$ are main-votes for b, and so all main-votes accepted by honest parties in round $r + 1$ will be main-votes for b. □

Agreement follows from part (d). All that remains is *termination*. For this, we need to show *deadlock freeness* and *fast convergence*.

Deadlock freeness is fairly straightforward. It is clear that honest parties will proceed from one round to the next, provided the adversary delivers enough messages between the honest parties. The *deadlock freeness* property follows from this observation, along with part (d) of Lemma 2, and the fact that parties who decide play along for one more round.

All that remains is *fast convergence.* Lemma 2 says that in a given round $r+1$, for $r \geq 1$, the set of $n-t$ main-votes accepted by an honest party in step 3 contains votes for either 0 or 1, but not both. Also, such an honest party will decide in this round unless it accepts at least one main-vote of `abstain`. But if it does accept an `abstain`, then $\rho_r = 1 - F(TID, r)$. The key to showing fast termination will be to show that *the value of ρ_r is determined before the coin (TID, r) is revealed.*

By "ρ_r is determined at a particular point in time," we mean the following: There is an efficient procedure W that takes as input a transcript describing the adversary's interaction with the system up to the given point in time, along with TID and $r \geq 1$, and outputs $w \in \{0, 1, ?\}$. Furthermore, if the output is $w \neq ?$, then if ρ_r ever becomes defined, it must be equal to w (or at least, it should be computationally infeasible for an adversary to cause this not to happen).

By "the coin (TID, r) is revealed at a particular point in time," we mean the point in time when an honest party generates the $(n-2t)$-th share of the coin (TID, r).

Lemma 3 *There is a function W that determines ρ_r, as described above, such that for all $r \geq 1$, either ρ_r is determined before coin (TID, r) is revealed, or ρ_{r+1} is determined before $(TID, r+1)$ is revealed.*

Proof. Suppose an honest party P_i is just about to generate the $(n-2t)$-th share of coin (TID, r) in step 1 of round $r+1$. As such, there is a set $\mathcal{S}$ of at least $n-2t$ honest parties who have also reached step 1 of round $r+1$; this set includes P_i, who is just about to release its share; all other members of $\mathcal{S}$ have already released their share. Almost all round $r+1$ pre-votes for the parties in $\mathcal{S}$, as well as their justifications, are completely determined at this point, even if these votes have not actually been cast. The only exception are soft pre-votes, whose actual value is equal to $F(TID, r)$, which is not yet known.

If any party in $\mathcal{S}$ is going to cast a hard pre-vote for $b \in \{0, 1\}$, then by Part (b) of Lemma 2, b is the only possible value for ρ_r. Thus, ρ_r is already determined—in fact, it is already defined.

Otherwise, all parties in $\mathcal{S}$ are going to cast soft pre-votes, choosing the value $F(TID, r)$ as the value of their round $r+1$ pre-vote. It follows that the only possible value for ρ_{r+1} is $F(TID, r)$. Therefore, immediately *after* P_i reveals its share of coin (TID, r), ρ_{r+1} is determined. Moreover, the coin $(TID, r+1)$ has not yet been revealed at this point, since fewer than $n-2t$ honest parties have gone beyond step 2 of round $r+1$. Thus, ρ_{r+1} is determined before coin $(TID, r+1)$ is revealed. □

This lemma, together with the unpredictability property of sequences of coins described in §4.3, implies that the probability that any honest party advances more than $2r+1$ rounds is bounded by $2^{-r} + \epsilon$, where ϵ is negligible. *Fast convergence* follows immediately. Note that to make this argument rigorous, we need to be able to explicitly "predict" (as in §4.3) the desired value of the coin $(1 - \rho_r)$ that would delay termination, which is why we defined the notion of "determining" ρ_r as we did.

We remark that if the first honest parties to decide make their decision in round r, there may be others who make their decision in round $r+1$. The "early deciders" play along for round $r+1$, which allows the "late deciders" to decide. However, the "late deciders" do not "know" they are "late," so they attempt to play along for round $r+2$. What happens is that in round $r+2$, the protocol will "fizzle out": the "late deciders" will simply end up waiting in step 2 for $n-t$ messages that never arrive. This "fizzling out" does indeed satisfy our technical definition of termination, and is perhaps adequate for some settings; however, a more "decisive" termination can be achieved with a minor modification of the protocol (see §5.3.1).

That completes the proof of *agreement* and *termination* for the case $f = t$. We now sketch the differences for the case $f < t$. There are some annoying technical problems that arise in this case because there is a gap between the number $(n - 2t)$ of shares for a signature (or coin) that need to be revealed before the signature (or coin) *may* be reconstructible, and the number $(n - t - f)$ of shares that need to be revealed before it *can* be reconstructed. We could have defined security for threshold signatures (coins) so that this gap did not exist; however, such a definition would be stronger than necessary.

Consider an adversary that chooses to corrupt a set $\mathcal{C}$ of $f < t$ parties. Let $\mathcal{H}$ denote the set of $n-f$ honest parties. We choose an arbitrary subset $\mathcal{Q} \subset \mathcal{H}$ of $t-f$ "quasi-corrupted" parties. The idea is that for the purposes of *agreement* and *termination*, parties in $\mathcal{Q}$ are considered to be honest, but for the purposes of the threshold signature and coin-tossing schemes, parties in $\mathcal{Q}$ are considered corrupted.

What this means concretely is that for parties in $\mathcal{Q}$, their secret shares for the threshold schemes are revealed to the adversary, but they otherwise behave as honest players with which the adversary interacts in the usual way. The main implication of this is that a particular signature or coin can be reconstructed if and only if at least $n - 2t$ parties in $\mathcal{H} \backslash \mathcal{Q}$ contribute shares. We also modify the proof as follows:

- In formulating the definition of ρ_r, we only count votes cast by members of $\mathcal{H} \backslash \mathcal{Q}$.
- In formulating the notion of precisely when a coin is revealed, we only count shares generated by parties in $\mathcal{H} \backslash \mathcal{Q}$.

With these modifications, Lemmas 2 and 3 can easily be proved, exactly as stated, and from these, *agreement* and *termination* follow.

5.3 Variations

Protocol ABBA can be modified several ways.

5.3.1 Achieving Stronger Termination

As we briefly discussed in §5.2, some parties may terminate an instance of a protocol in a rather indecisive way: although they have made a decision, they do not know that they can stop; instead, they will simply block, waiting forever for messages that will never arrive. It is not clear to what extent this is a serious problem, but anyway, it is easy to modify protocol ABBA so that parties not only decide, but terminate in a more decisive fashion. Namely, when a party P_i decides b for *TID* in round r, it can combine the signature shares that it has on hand to construct an $\mathcal{S}$-threshold signature on the message

$$(\texttt{main-vote}, \mathit{TID}, r, b).$$

It then sends this threshold signature to all parties and stops. Thus, P_i can effectively erase all data in its internal state relevant to *TID*, and ignore all future incoming messages relating to *TID*. Any other party that is waiting for some other message, but instead receives the above threshold signature, can also decide b for *TID*, send the this signature to all parties, and then stop.

Note that without this modification, the threshold signatures on main-votes other than `abstain` are actually not used by the protocol, and could be deleted.

5.3.2 Using an $(n, t+1, t)$ Coin-Tossing Scheme

Instead of an $(n, n-t, t)$ coin-tossing scheme, one could use an $(n, t+1, t)$ coin-tossing scheme, provided that before a party releases its share of a coin, it sends an appropriate "ready" message to all parties, and waits for $n-t$ corresponding "ready" messages from other parties. These "ready" messages do not need to be signed—the authenticity of the messages is enough. This modification increases the communication complexity of the protocol; however, an $(n, t+1, t)$ coin can be implemented based on weaker intractability assumptions than and $(n, n-t, t)$ coin, and so the tradeoff may be worthwhile in some settings.

5.3.3 Further Optimizations

Although we have strived to make our protocol as efficient as possible, we have omitted several optimizations in order to simplify the presentation; they are described next. Some of them lead to a more flexible, "pipelined" execution of the protocol steps.

1. A party need not generate a share of the coin in round $r+1$ if it did not accept a main-vote of `abstain` in round r.

2. A party need not wait for $n-t$ coin shares, unless it is going to cast a soft pre-vote, or unless it needs to later verify the justification of a soft pre-vote (it can always wait for them later if needed).

3. A party need not wait for $n-t$ pre-votes once it accepts two conflicting pre-votes, since then it is already in a position to cast a main-vote of `abstain`.

4. A party need not wait for $n-t$ main-votes if it has already accepted a main-vote for something other than `abstain`, since then it is already in a position to move to the next round; however, the decision condition should be checked before the end of the next round.

5. It is possible to collapse steps 4 and 1; however, some adjustments must be made to accommodate the threshold signature. If a party wants to make a hard pre-vote for b, he should generate signature shares on two messages that say "I pre-vote b if the coin is 0" and "I pre-vote b if the coin is 1." If a party wants to make a soft pre-vote, he should generate signature shares on two messages that say "I pre-vote 0 if the coin is 0" and "I pre-vote 1 if the coin is 1." This allows the parties to make soft pre-votes and reveal the coin concurrently, while also making it possible to combine both soft and hard pre-votes for the same value to construct the necessary main-vote justifications. This variation reduces the round and message complexity by a factor of 1/3, at the expense of somewhat higher computational and bit complexity; it also precludes variations (1) and (2) above.

6 A Diffie-Hellman Based Threshold Coin-Tossing Scheme

6.1 The Scheme

In this section, we present an (n, k, t) threshold coin-tossing scheme based on the Diffie-Hellman problem. We work with a group G of large prime order q.

At a high level, our scheme works as follows. The value of a coin C is obtained by first hashing C to obtain $\tilde{g} \in G$, then raising $\tilde{g}$ to a secret exponent $x_0 \in \mathbb{Z}_q$ to obtain $\tilde{g}_0 \in G$, and finally hashing $\tilde{g}_0$ to obtain the value $F(C) \in \{0, 1\}$. The secret exponent x_0 is distributed among the parties using Shamir's secret sharing scheme [35]. Each party P_i holds a share x_i of x_0; its share of $F(C)$ is $\tilde{g}^{x_i}$, along with a "validity proof." Shares of coin C can then be combined to obtain $\tilde{g}_0$ by interpolation "in the exponent."

In more detail, we need cryptographic hash functions

$$\begin{aligned} H &: \{0,1\}^* \to G; \\ H' &: G^6 \to \mathbb{Z}_q; \\ H'' &: G \to \{0,1\}. \end{aligned}$$

No specific requirements are made for these hash functions, but they will be modeled as random oracles in the analysis. (H'' could actually be implemented in the standard model, e.g., by the inner product of the bit representation of the input with a random bit string, chosen once and for all by the dealer.)

In the dealing phase, the dealer selects k coefficients of a random polynomial $f(T)$ over $\mathbb{Z}_q$ of degree less than k and a random generator g of G. For $0 \leq i \leq n$, let $x_i = f(i)$ and $g_i = g^{x_i}$. Party P_i's secret key SK_i is x_i, and his verification key VK_i is g_i. The global verification key VK consists of a description of G (which includes q) and g.

For a general coin $C \in \{0,1\}^*$, we let $\tilde{g} = H(C)$, and $\tilde{g}_i = \tilde{g}^{x_i}$ for $0 \leq i \leq n$. The value of the coin is $F(C) = H''(\tilde{g}_0)$.

For a given coin C, party P_i's share of the coin is $\tilde{g}_i$, together with a "validity proof," i.e., a proof that $\log_{\tilde{g}} \tilde{g}_i = \log_g g_i$. This proof is the well-known interactive proof of equality of discrete logarithms (see [14]), collapsed into a non-interactive proof using the Fiat-Shamir heuristic [22]. A valid proof is a pair $(c, z) \in \mathbb{Z}_q \times \mathbb{Z}_q$, such that

$$c = H'(g, g_i, h, \tilde{g}, \tilde{g}_i, \tilde{h}), \tag{1}$$

where

$$h = g^z / g_i^c \text{ and } \tilde{h} = \tilde{g}^z / \tilde{g}_i^c.$$

Party P_i computes such a proof by choosing $s \in \mathbb{Z}_q$ at random, computing $h = g^s$, $\tilde{h} = \tilde{g}^s$, and obtaining c as in (1) and $z = s + x_i c$.

Now, for any set S of k distinct points in $\mathbb{Z}_q$, and any $\beta \in \mathbb{Z}_q$, there exist elements $\lambda^S_{\alpha,\beta} \in \mathbb{Z}_q$ for $\alpha \in S$, such that

$$\sum_{\alpha \in S} f(\alpha) \lambda^S_{\alpha,\beta} = f(\beta).$$

These λ-values are independent of $f(T)$, and can be computed from the formulas for Lagrange interpolation.

To combine a set of valid shares $\{\tilde{g}_\alpha : \alpha \in S\}$, one simply computes

$$\tilde{g}_0 = \prod_{\alpha \in S} \tilde{g}_\alpha^{\lambda^S_{\alpha,0}}.$$

The value of the coin is then computed as $H''(\tilde{g}_0)$.

6.2 Security Analysis

To analyze this scheme, we need to consider the following two intractability assumptions. For $g, g_0, \hat{g} \in G$, define $DH(g, g_0, \hat{g})$ to be $\hat{g}_0 = \hat{g}^{x_0}$, provided that $g_0 = g^{x_0}$. Also, define $DHP(g, g_0, \hat{g}, \hat{g}_0)$ to be 1 if $\hat{g}_0 = DH(g, g_0, \hat{g})$, and 0 otherwise.

The Computational Diffie-Hellman (CDH) assumption is the assumption that *DH* is hard to compute—that is, there is no efficient, probabilistic algorithm that computes *DH* correctly (with negligible error probability) on all inputs.

The Decisional Diffie-Hellman (DDH) assumption is the assumption that *DHP* is hard to compute—that is, there is no efficient, probabilistic algorithm that computes *DHP* correctly (with negligible error probability) on all inputs.

Theorem 4 *In the random oracle model, the above coin-tossing scheme is secure under the CDH assumption, if $k = t + 1$, and under the DDH assumption otherwise.*

We need to show *robustness* and *unpredictability.*

The robustness of the scheme follows from the soundness of the interactive proof of equality of discrete logarithms, and the fact that in the random oracle model, the challenges c are the output of the random oracle H'.

To prove unpredictability, we assume we have an adversary that can predict a coin with non-negligible probability, and show how to use this adversary to efficiently compute *DH* (if $k = t + 1$) or *DHP* (if $k > t + 1$).

We first make a few simplifying assumptions:

- the adversary corrupts parties $P_{k-t}, \ldots, P_{k-1}$;
- before the adversary requests the share of a coin or predicts a coin, he has already evaluated H at that coin's name;
- the adversary evaluates H successively at distinct points $C_1, \ldots, C_l$, where l is a bound that is fixed for a given adversary and security parameter.

We denote the "target" coin, which the adversary attempts to predict, by $\hat{C}$, and we let $\hat{g} = H(\hat{C})$, and $\hat{g}_i = \hat{g}^{x_i}$ for $0 \leq i \leq n$.

We may assume that $\hat{C}$ is equal to C_s, where s is randomly chosen from $\{1, \ldots, l\}$. Should the adversary makes $k - t$ requests to reveal shares of $\hat{C}$, we simply stop the game. This decreases the adversary's advantage by a factor of l.

Case 1: $k = t + 1$. Here is how we use this adversary to compute *DH*. By the results of Shoup [36], it is sufficient to construct an algorithm that on random inputs $g, g_0, \hat{g} \in G$, outputs a list of group elements that contains $\hat{g}_0 = DH(g, g_0, \hat{g})$ with non-negligible probability.

We simulate the adversary's interaction with the coin-tossing scheme as follows. By our simplifying assumption, the adversary corrupts parties $P_1, \ldots, P_t$. As the notation suggests, we use the given value g in the global verification key. We choose $x_1, \ldots, x_t \in \mathbb{Z}_q$ at random, set $S = \{0, 1, \ldots, t\}$, compute $g_i = g^{x_i}$ for $1 \leq i \leq t$, and let for $t + 1 \leq i \leq n$

$$g_i = \prod_{j=0}^{k-1} g_j^{\lambda_{j,i}^S}.$$

In the random oracle model, the adversary explicitly queries the random oracles H, H', H''. The simulator we are building is responsible for the operation of these oracles—it sees the queries made by the adversary, and is free to respond as it wishes so long as its responses are consistent and correctly distributed. As the notation suggests, we use the given $\hat{g}$ as the value of H at $\hat{C}$ (whatever $\hat{C}$ turns out to be).

For coins $C \neq \hat{C}$, we choose $r \in \mathbb{Z}_q$ at random and compute $\tilde{g} = g^r$. The simulator uses the given $\tilde{g}$ as the value of H at C. We then compute the shares $\tilde{g}_i = g_i^r$ for $t+1 \leq i \leq n$. The validity proofs can be simulated using standard zero-knowledge techniques [24].

For the target coin $\hat{C}$, we never have to compute any shares for honest parties, since $k = t+1$. When the adversary terminates, we simply output the list of queries made by the adversary to the oracle H''.

It is easily verified that the above simulation is nearly perfect: the adversary's view has precisely the same distribution as in the actual interaction (but there is actually a negligible probability that the zero-knowledge simulations fail).

Observe that because the adversary has a non-negligible advantage in predicting the value of the coin $\hat{C}$, he must evaluate H'' at the corresponding point $\hat{g}_0$ with non-negligible probability. That completes the proof of Theorem 4 for Case 1.

Case 2: $k > t+1$. The above simulation does not work in this case because we would have to simulate the shares of the coin $\hat{C}$ from up to $k-t-1 > 0$ honest parties. Moreover, we cannot view these honest parties as fixed: the adversary may adaptively select which honest parties contribute shares of the target coin. So instead, in this case, we use the adversary to compute *DHP*. Actually, it is sufficient [39, 31] to construct a statistical test that distinguishes between the following two distributions

D: the set of tuples

$$(g, g_0, \ldots, g_{k-t-1}, \hat{g}, \hat{g}_0, \ldots, \hat{g}_{k-t-1}),$$

where $g, g_0, \ldots, g_{k-t-1} \in G$ are random, and $\hat{g} = g^r, \hat{g}_0 = g_0^r, \ldots, \hat{g}_{k-t-1} = g_{k-t-1}^r$ for randomly chosen $r \in \mathbb{Z}_q$; and

R: the set of tuples

$$(g, g_0, \ldots, g_{k-t-1}, \hat{g}, \hat{g}_0, \ldots, \hat{g}_{k-t-1}),$$

where $g, g_0, \ldots, g_{k-t-1}, \hat{g}_0, \ldots, \hat{g}_{k-t-1} \in G$ are random.

Our statistical test works as follows. Let

$$(g, g_0, \ldots, g_{k-t-1}, \hat{g}, \hat{g}_0, \ldots, \hat{g}_{k-t-1})$$

be the input "test" tuple. We simulate the adversary's interaction with the coin-tossing scheme as follows. By our simplifying assumption, the adversary corrupts $P_{k-t}, \ldots, P_{k-1}$. As the notation suggests, we simulate the dealer by using the given g in the global verification key, and $g_1, \ldots, g_{k-t-1}$ in the local verification keys for $P_1, \ldots, P_{k-t-1}$. We choose the secret keys $x_{k-t}, \ldots, x_{k-1} \in \mathbb{Z}_q$ at random and set $S = \{0, 1, \ldots, k-1\}$; for $k-t \leq i \leq k-1$, compute $g_i = g^{x_i}$, and for $k \leq i \leq n$, let

$$g_i = \prod_{j=0}^{k-1} g_j^{\lambda_{j,i}^S}.$$

Also, we will use the given $\hat{g}$ as the output of H at $\hat{C}$, and the given $\hat{g}_1, \ldots, \hat{g}_{k-t-1}$ as the corresponding shares of $\hat{C}$ from parties $P_1, \ldots, P_{k-t-1}$. We will use the given $\hat{g}_0$ to compute the shares of $\hat{C}$ from the other honest parties as follows: for $k-t \leq i \leq k-1$, set $\hat{g}_i = \hat{g}^{x_i}$, and for $k \leq i \leq n$, compute

$$\hat{g}_i = \prod_{j=0}^{k-1} \hat{g}_j^{\lambda_{j,i}^S}.$$

Whenever the adversary requests a share of $\hat{C}$ for an honest party P_i, we give the adversary $\hat{g}_i$ as computed above.

We reveal the shares of a coin $C \neq \hat{C}$ just as in Case 1: we choose $r \in \mathbb{Z}_q$ at random, and compute $\tilde{g} = g^r$ and $\tilde{g}_i = g_i^r$ for all $1 \leq i < k - t$ and $k \leq i \leq n$.

For both target and non-target coins, we construct simulated proofs of correctness just as in Case 1.

At the end of the adversary's interaction, when the adversary makes a prediction $b \in \{0, 1\}$ for the value of coin $\hat{C}$, we output $X = 1$ if $b = H''(\hat{g}_0)$, and $X = 0$ otherwise.

We claim that this algorithm is an effective statistical test distinguishing **D** from **R**.

Observe that if the test tuple comes from **D**, the above simulation is nearly perfect, and so the probability that $X = 1$ is essentially the adversary's advantage, which differs from 1/2 by a non-negligible amount.

Therefore, it will suffice to show that if the test tuple comes from **R**, the probability that $X = 1$ differs from 1/2 by a negligible amount. But this follows from the observation that for any sequence of distinct indices $i_1, \ldots, i_{k-t-1}$ belonging to honest parties, the group elements

$$\hat{g}_0, \hat{g}_{i_1}, \ldots, \hat{g}_{i_{k-t-1}}$$

are independent and uniformly distributed. Thus, after revealing any $k - t - 1$ of the "shares" $\hat{g}_i$ belonging to honest parties, then conditioning on the adversary's view, the value of $\hat{g}_0$ is still random, and hence the probability that $X = 1$ in this case is essentially 1/2.

This completes the proof of Theorem 4 for Case 2. Note that in the proof of this, we do not need to model H'' as a random oracle—we only need the property that for random $\hat{g}_0 \in G$, $H''(\hat{g}_0)$ has a nearly uniform distribution. For example, using the Entropy Smoothing Theorem [27, Chapter 8], one could implement H'' as the inner product of the bit representation of $\hat{g}_0$ with a random bit string (chosen once and for all by the dealer). Also note that using the same proof technique, one could prove the unpredictability property using the threshold $k - f$ instead of $k - t$, where f is the actual number of corrupted parties.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. To appear in *IEEE Journal on Selected Areas in Communication*. Revised version of IBM Research Report RZ 2976 and extended abstracts in *Proc. Eurocrypt '98* and *Proc. IEEE Symp. on Security and Privacy (1998)*, Oct. 1999.

[2] D. Beaver and N. So. Global, unpredictable bit generation without broadcast. In T. Helleseth, editor, *Advances in Cryptology: EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*. Springer, 1994.

[3] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, 1998.

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM Conference on Computer and Communications Security*, 1993.

[5] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *Proc. 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 57–66, 1995.

[6] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC)*, 1983.

[7] P. Berman and J. A. Garay. Randomized distributed agreement revisited. In *Proc. 23th International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 412–419, 1993.

[8] C. Boyd. Public-key cryptography and re-usable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, 1989.

[9] G. Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–162, 1984.

[10] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, 1996.

[11] R. Canetti and S. Goldwasser. An efficient threshold public-key cryptosystem secure against adaptive chosen-ciphertext attack. In J. Stern, editor, *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999.

[12] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 42–51, 1993.

[13] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[14] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology: CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.

[15] B. Chor and C. Dwork. Randomization in Byzantine agreement. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 443–497. JAI Press, 1989.

[16] R. A. Croft and S. P. Harris. Public-key cryptography and re-usable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 189–201. Clarendon Press, 1989.

[17] Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *Advances in Cryptology: CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer, 1988.

[18] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology: CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1990.

[19] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, Oct. 1990.

[20] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. In J. Hlavicka, E. Maehle, and A. Pataricza, editors, *Proc. 3rd European Dependable Computing Conference (EDCC-3)*, volume 1667 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 1999.

[21] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, Aug. 1997.

[22] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology: CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1987.

[23] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[24] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, Feb. 1989.

[25] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988.

[26] K. Kursawe. Optimistic asynchronous Byzantine agreement. Manuscript, 1999.

[27] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.

[28] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, 1996.

[29] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. In D. Coppersmith, editor, *Advances in Cryptology: CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 185–196. Springer, 1995.

[30] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In J. Stern, editor, *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*. Springer, 1999.

[31] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.

[32] M. O. Rabin. Randomized Byzantine generals. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 403–409, 1983.

[33] T. Rabin. A simplified approach to threshold and proactive RSA. In H. Krawczyk, editor, *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[34] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, Jan. 1996.

[35] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[36] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology: EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*. Springer, 1997.

[37] V. Shoup. On formal models for secure key exchange. Research Report RZ 3120, IBM Research, 1999. revision 4, available from `http://www.shoup.net`.

[38] V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Advances in Cryptology: EUROCRYPT 2000*, Lecture Notes in Computer Science. Springer, 2000. (to appear).

[39] M. Stadler. Publicly verifiable secret sharing. In U. Maurer, editor, *Advances in Cryptology: EUROCRYPT '96*, volume 1233 of *Lecture Notes in Computer Science*, pages 190–199. Springer, 1996.

[40] S. Toueg. Randomized Byzantine agreements. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 163–178, 1984.

[41] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, Apr. 1996.

[42] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.