Maintaining Authenticated Communication in the Presence of Break-ins

Ran Canetti* — IBM T.J. Watson Research Center

Shai Halevi† — MIT, Lab for Computer Science

Amir Herzberg‡ — IBM Haifa Research Lab - Tel Aviv Annex

Abstract

We study the problem of maintaining authenticated communication over untrusted communication channels, in a scenario where the communicating parties may be occasionally and repeatedly broken into for limited periods of time. Once a party is broken into, its cryptographic keys are exposed and perhaps modified. We describe a mechanism that allows a party whose security has been compromised to *regain* its ability to communicate in an authenticated way.

The contribution of this paper is twofold. First we present a mathematical model for analyzing this scenario, and exhibit various properties and parameters of this model. Next we describe a practically-appealing protocol which enables parties to maintain authenticated communication in the presence of such a powerful adversary. For this protocol we use a variation of the proactive distributed signature schemes which were recently described by Herzberg et al. Although these schemes are designed for a model where authenticated communication and broadcast primitives are available, we show how they can be modified to work in our model, where no such primitives are available a-priori. We also present a new proactive distributed signature scheme with improved round and communication complexities.

1 Introduction

Maintaining authenticated communication over an untrusted network is one of the most basic goals in cryptography. Practically no cryptographic application can get off the ground without authenticity, and once authenticity is achieved other cryptographic goals (such as secrecy) are achieved using standard tools.

Authenticated communication can be maintained using a variety of standard cryptographic techniques, even when the communication links are fully controlled by an adversary, provided that some cryptographic keys are distributed ahead of time. Yet, all cryptographic techniques rely on the ability of the communicating parties to maintain the integrity and secrecy of their cryptographic keys. Indeed, attacks on the security of the communication systems are often based on obtaining, or even modifying, these keys, through system penetration (rather than cryptanalysis). We call such attacks break-in attacks. How can authenticated communication be maintained in the presence of repeated break-in attacks? In particular, how can a party recover from a break-in and regain its security? Answering these questions is the focus of our paper.

Contributions of This Paper. This paper consists of three parts which are, to some extent, independent of each other. In the first part (Section 2) we devise a formal model for analyzing the problem of authenticated communication in the presence of break-in attacks and exhibit the fundamental properties and parameters of this model. In the second part (Sections 4 through 6) we describe a protocol for solving this problem and analyze its properties. The main tool in this protocol are proactive signatures schemes (e.g., [18, 16, 13, 12, 14]). In particular, we show how proactive signatures can be done even when the links are not authenticated. In the third part (Section 7) we describe another proactive signatures scheme with improved round and communication complexities.

1.1 Fundamentals of the Problem

We consider a network of nodes that communicate via "totally unreliable" communication links. This is modeled by assuming that an adversary has full control over the links. That is, the adversary can read, modify, delete and duplicate messages sent over the links, or even inject its own messages. Break ins are modeled by allowing the adversary to occasionally break into nodes and obtain all their secret data; even further, this secret data may be modified by the adversary. In the presence of such an adversary, we seek a protocol that enables the nodes to recover and maintain authenticated communication.

Recovery from break-ins. First we note that if we are to offer any solution to this problem, we should make sure that the adversary cannot modify the code of the protocol itself (otherwise there is no hope for recovery). Therefore we assume that the program of each node is written in a Read-Only-Memory (ROM) and cannot be modified by anyone. (See discussion on the ROM assumption at the end of the Introduction.)

Also, it is clear that no protocol can guarantee authenticated communication if the adversary breaks into all the nodes in the network *at the same time*. Therefore, we must put some limitations on the power of the adversary in order to make a solution possible; the quality of a solution will be measured by the "strongest adversary" it can withstand, where the power of an adversary is measured as follows: Following the proactive approach in [21, 6, 17] we introduce periodical, short refreshment phases, during which the nodes jointly try to refresh their keys and regain security. Let a time unit denote the time between two consecutive refreshment

*POB 704, Yorktown Heights, NY 10598, canetti@watson.ibm.com. Part of this research was done while in the Laboratory of Computer Science in MIT.

†NE43-342, 545 Technology Square, Cambridge MA 02139, shaih@theory.lcs.mit.edu

‡IBM, 2 Weizmann st., Tel-Aviv, Israel, amirh@vnet.ibm.com

phases. The power of an adversary is measured by the number of nodes broken into at any single time unit and by the number (and connectivity) of communication links where messages are modified in any time unit.

Unavoidable attacks. Let us gain more intuition for the problems this model suggests, by describing some unavoidable attacks. The first attack is trivial: the adversary may fail to deliver messages. Hence we can never guarantee that the computation will proceed as if the links were reliable (e.g., in case of a network partition). Authentication protocols therefore aim for the weaker requirement that only authentic messages be accepted as such. Namely, they require that *a node will not accept an incoming message which was modified (or injected) by the adversary*.

However, it turned out that in the presence of break-ins, even this weaker requirement is sometime impossible to achieve. To see this, consider a node N that was recently broken into, so the adversary knows N's "secret" cryptographic keys. The adversary, controlling the communication, can now 'cut off' N from the rest of the nodes, and impersonate N in their eyes. The impersonation can be kept up *even after N is no longer broken into,* and no node (other than N itself) can ever notice this attack. These impersonation attacks too are unavoidable in our model. The most which we can expect from a protocol in this model, is that the impersonated nodes will be aware of their situation, and that the adversary should be required to "work hard" in order to impersonate a node (thus, an adversary with limited resources cannot impersonate too many nodes).

1.2 Defining our Goals

In light of the above unavoidable attacks, one may wonder if there is any meaningful way to define a "solution" for the problem of authenticated communication in this model. Indeed, formulating a reasonable notion of a solution is one of the main contributions of this paper. Below we very roughly sketch the definition. (See more precise definition later.)

Requirements from a protocol. We want to devise a general tool that enables protocols to run on an unauthenticated network with break-ins 'as if' it was an authenticated network. In other words, we want to construct a 'compiler' that transforms protocols designed for authenticated networks into ones which run over unauthenticated networks in the presence of break-ins, while maintaining their functionality as much as possible. In Section 2 we define two complementary notions of maintaining functionality, called resilience and awareness.

Informally, we say that a node N is being impersonated if some other node M accepts a message as being sent from N while N did not send this exact message to M at this round. Awareness now means that an impersonated node is aware of its situation. (This awareness can be translated into a special `alert` signal which the impersonated node outputs to a higher layer protocol or an operator.) Resilience means that it is hard to impersonate nodes. More precisely, a resilient protocol should guarantee that no node is impersonated, unless it is either broken into, or is disconnected from a large fraction of the network. (See a formal definition within.)

In a sense, awareness is a correctness property while resilience is a liveness property: The former requires that some bad situation (i.e., when a node is impersonated without knowing about it) never occurs. The later means that nodes are not being impersonated (banning some powerful attack by the adversary) and therefore the computation of the protocol can continue.

We remark that the combination of awareness and resilience imply other desirable properties of protocols in our model, such as the ability to show (using the framework of secure multiparty computation) that running the modified protocol in our model is as secure as running the original one in an *authenticated network* where the only possible link failures are omission of messages (so the adversary cannot modify or inject messages).

The power of the adversary. At any given time during the protocol, we distinguish between three categories of nodes: A node can be either broken into, 'disconnected' from a large fraction of the other nodes, or else it is 'operational'. The adversary's power is thus captured using tree parameters: the number of nodes broken into at any given time unit, the number (and connectivity) of links that have to be tampered with in order to 'disconnect' a node, and the number of disconnected nodes. See details within.

Related definitions. The problem which we consider in this work is related to the problem of entity and session authentication [1, 4, 2]. The main differences are that in these works, nodes are identified by virtue of knowing some cryptographic keys which remain unknown to the adversary throughout. This is clearly impossible in our model. On the other hand, these works emphasize the ability to run multiple authenticated sessions at the same time, while our model does not address this problem.

1.3 Towards a solution.

Consider the following first attempt at a solution: At system set-up, each node gets a private signature-key of some signature scheme, along with the verification keys of the other nodes. Authentication is now straightforward: each node signs each outgoing message, and a received message is accepted only if verification of the sender's signature succeeds. (Alternatively, nodes can exchange symmetric session keys and use them to authenticate messages). Since the nodes can be periodically broken into, a first step towards a solution will thus be to let each node replace, at each refreshment phase, its old keys with a freshly chosen pair of signature and verification keys. This way, the new signature key remains secret as long as the node is not broken-into again.

The crux of the problem now is how to make sure that other nodes learn the new verification key. This is a non-trivial task: consider a node N just recovering from a break-in. N's old signature key is compromised. Thus, the adversary can forge N's signature and send a fake new verification key in the name of N, thus successfully impersonating N. Indeed, as explained above, this kind of attack is unavoidable in this model, but we want to make sure that N is *aware* of the fact that the adversary is impersonating it.

One may attempt to obtain awareness using signed "echos": Let each node M acknowledge N's new verification key back to N and sign the acknowledgment using M's signature key, so N can verify that M got its new public key. Indeed, such a solution may work if N still has M's verification key. However, the adversary can erase M's verification key in N's memory, and plant a fake key (for which the adversary knows the corresponding signature key) in its place. The adversary can now impersonate N in the eyes of M (and, in fact, in the eyes of all other nodes) and at the same time impersonate all other nodes in the eyes of N!

This brings us to the main technical difficulty in our model. Namely, we need to have a way of "bootstrapping trust" during refreshment phases, *when a node cannot even trust its own memory*. We get around this seemingly hopeless situation by letting each node keep a small, unchanging public key in an unmodifiable ROM together with the code of the protocol, and use this key to bootstrap trust. We outline our solution below.

1.4 Our Solution

The main technical tool in our solution is a proactive distributed signature scheme (PDS). Distributed signature schemes can be very roughly described as follows. Each node has its share of a private signature key, in a way that enables any large enough subset of the nodes to jointly generate a valid, unforgeable signature of a given message, while preventing the adversary from generating valid signatures. Signatures can be verified using a public verification key. In a proactive distributed signature scheme, the nodes refresh their shares of the private signature key at each refreshment phase. This is done in a way that prevents the adversary from generating valid signatures as long as not too many nodes are broken into between two consecutive refreshment phases. We stress that, even though the shares of the signature key are changed periodically, the public verification key *remains unchanged throughout.*

In our solution each node keeps a copy of the unchanging, public verification key of a PDS scheme in a small ROM. Thus, a node is always able to verify (and trust) signatures with respect to this key. Such signatures can be generated jointly by the nodes. Authenticated communication is now achieved as follows. At each refreshment phase each node executes the following protocol:

- Choose a new set of 'personal' signature and verification keys of some 'standard' (non-distributed) signature scheme.
- Obtain a certificate, generated jointly by all nodes using the PDS scheme, for the new verification key. (The certificate may read: *"it is certified that the personal verification key of N_i for time unit ℓ is v".*)

  The PDS scheme ensures that at most one certificate is created for every node at every time unit. (Hence, if N_i obtains a certificate in time unit ℓ, it is guaranteed that the adversary can not obtain a fake certificate with N_i's name on it in this round.) We note that a single instance of the PDS scheme is used to generate the certificates of all the nodes.
- Next, some work is done in order to maintain the PDS scheme itself.

For the rest of the time unit, the obtained certificates are used in the standard way: That is, each node signs each of its messages using its current signature key, and attaches the corresponding verification key along with the newly obtained certificate. The recipient of a message verifies the certificate using the verification key in its secure ROM, and then verifies the sender's signature of the messages (Alternatively, the nodes can use the certificates to exchange a shared key for the rest of the time unit, and use the shared key to authenticate messages.)

If a node fails to obtain a certificate for its new verification key, or if it fails to refresh its share of the PDS scheme, then this node issues an `alert` signal which signals that this node is under attack by the adversary, and may not be able to authenticate its communication. This alert may be handled out-of-band by a higher level protocol or the operator.

We show that this simple-to-implement construction guarantees awareness and resilience as long as the adversary does not break-into or disconnect more than half of the nodes during a single time unit.

Realizing Proactive Signatures over Untrusted Links. For our construction we use proactive signature schemes, such as the ones described in [18, 16, 13, 14]. All these schemes, however, require authenticated broadcast channels, whereas in our model the channels are unauthenticated. We therefore extend the above works by describing how these schemes can be implemented over unauthenticated channels. For this purpose we use standard Byzantine Agreement tools designed for authenticated networks [22, 19, 20, 7, 11], as well as some additional simple protocols. In our scenario these tools guarantee a somewhat weaker variant of broadcast; this variant will suffice for our needs. Roughly speaking, our protocol ensures the existence of a large enough 'clique' of nodes, among which there exists a virtual authenticated broadcast channel.

We also describe, in Section 7, another proactive signature scheme, which has some nice on-line/off-line tradeoff (cf. [8]). This enables us to carry the bulk of the communication off-line, and to send only few messages during the refreshment phases.

1.5 Discussion

Scalability issues. To use our protocol in a very large network (e.g., the Internet), it is possible to partition the network to local neighborhoods, and then to perform the protocol only in local neighborhoods, with each neighborhood run its own PDS scheme. A node will obtain a certificate from one or more of the neighborhoods it belongs to. (The unchanging verification keys of all the neighborhoods can be signed — at system start-up — by a global certification authority.) The partition will impact the resilience as well as the performance, and the designer should pick a partition that offers a good tradeoff. For many practical scenarios, a partition to $\sqrt{n}$ clusters of $\sqrt{n}$ nodes would give good tradeoff (with resilience dropping to $n/4$). We believe however that there exist other refinements to the scheme that will handle scalability even better than partitions, at least in asymptotic terms.

Model assumptions - ROM and alert. Our model includes two essential, but non-standard, assumptions: the existence of ROM memory that the attacker can read but cannot modify, and the fact that the attacker can impersonate a node with the only penalty of an alert signal by that node. However we believe both assumptions are acceptable in most real-life applications.

In particular, the ROM requirement can be implemented in a variety of ways, and corresponds neatly to the well-accepted mechanisms for virus detection and removal. Similarly, in the practice of computer security and protection against hackers, it is well accepted that once a penetration is detected, the system managers and security officers are well equipped to regain control and expose the hackers. Therefore the threat of an alert is sufficient to deter the hackers.

In addition, with appropriate design of the operating system one can implement a 'virtual ROM' in software. That is, the OS can deny writing access to certain memory locations from *any* process, other than processes that are active only at start-up time. (For this to work, the code restricting access need also be written in the protected memory.)

2 Definitions

In this section we define the model of computation and the adversary, and formalize the requirements from a protocol for maintaining authenticated communication in this model.

2.1 Basics

We consider a synchronous network of n nodes, where the communication channels are unauthenticated and controlled by an adversary. This is modeled as follows. All nodes have access to a black-box representing the network. (We call this black-box the network.) At each communication round, each node N sends its messages by handing them to the network on a special `send` tape,

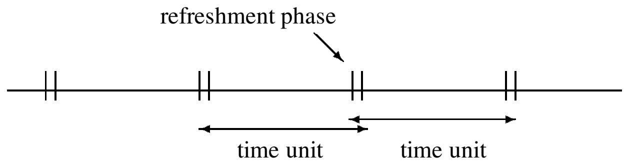

Figure 1: Time units and refreshment phases.

dedicated to N. Next N reads the messages delivered to him from the network in this round on a `receive` tape dedicated to N.[1]

The network is controlled by an adversary that can write anything on the nodes' receive tapes. On top of controlling the network, the adversary may also occasionally break into nodes during the course of the computation. Once a node is broken into, the entire contents of the node's current memory becomes known to the adversary. The adversary can also modify the memory contents of a broken node, except for its program. (We think of the program as being written in ROM.) However, at time units during which the node is broken into, it can be thought of as following the adversary's instructions. Once the adversary leaves a node it no longer can read or modify the node's internal data, and the node resumes executing its original program. (Of course, the adversary remembers data read in the past.)

Time Units and Refreshment Phases. We divide the lifetime of the system into time units. We let time units have small overlap; that is, a time unit starts slightly before the preceding time unit ends. We call the small overlap a refreshment phase. See Figure 1. Typically, a time unit may last several hours, days, weeks, months etc., whereas the duration of a refreshment phase is several communication rounds (i.e. up to a few seconds).

2.2 Compiling Protocols for an Unauthenticated Network

Our goal is to allow application protocols to run over an unauthenticated network 'as if' it consists of authenticated channels. Namely, we would like to devise a compiler for authenticated network, which transforms a protocol that is designed to run over a network of authenticated channels to one that can run over an unauthenticated network. (We stress that authenticated networks do not protect against message omissions, only against injection and modification.)

This compiler, denoted Λ, operate roughly as follows. Let Π be any protocol that assumes an authenticated network. We run Λ on Π (with security parameter k) to generate a protocol, denoted $\Lambda\Pi_k$, that can be thought of as having two layers. At the top layer Π runs unchanged. At the bottom layer, for each message that Π instructs to send and receive, the nodes will follow some procedure specified by Λ. We refer to this procedure informally as "security transformations". Also, in each refreshment phase the nodes execute some refreshment protocol specified by Λ. In the rest of this section we precisely define the requirements from such a compiler.

Computational Views and Impersonation. We next define the notion of impersonation attacks. To this end, consider a node N running an application protocol Π on top of Λ. We distinguish between the internal view of N and its external view. The internal view of N in time unit ℓ *in which N is not broken-into*, consists of all the messages which were sent and received by the application protocol Π which was running on N during this time unit. (Notice that since Π is running on top of Λ, then each message which is received by Π must have first been accepted by Λ.) If N was broken-into in this time unit then its internal view is undefined.

The external view of N in time unit ℓ consists of all the messages which appear, in the internal views of other nodes in the network, to be received from N. We say that a node N is being impersonated at time unit ℓ if its external view contains a message which is not in its internal view.(Note that being impersonated is a 'syntactic' property that does not depend on the structure of Π or Λ.

2.3 Measuring the Power of an Adversary

The adversary in our model has two capabilities: breaking into nodes and disrupting communication links. Accordingly, we characterize the power of the adversary by two parameters. Roughly, one parameter specifies the number of nodes the adversary can break-into in each time unit, and the other specifies the number of communication links it disrupts (on top of the links which are adjacent to nodes which are broken-into). To make this precise, we start with a definition of faithful links.

Definition 1 *We say that the link between nodes N_i and N_j is* faithful *during a certain time interval,*[2] *if during this time:*
(a) *Neither N_i nor N_j are broken into,*
(b) *Every message sent on this link by one of these nodes, arrives unmodified at the other node at the end of the communication round in which it was sent, and*
(c) *N_i, N_j do not receive any other message on this link.*

We proceed now to define the three categories of nodes in the network, namely broken, disconnected and operational nodes. As defined above, broken nodes are those broken into by the adversary at this time unit. Defining disconnected and operational nodes is more subtle. In principle, we would like to have a parameter t ($t \le n$, where n is the number of nodes in the network), so that an unbroken node is t-operational if it has at most t links which are not faithful, and is disconnected otherwise. Formalizing this notion in a meaningful way requires some care, however.

To see the problem with the straightforward formulation, consider a set of t nodes which were broken in time unit $(\ell - 1)$ but are not broken in time unit ℓ, and which have faithful links to one another during time unit ℓ. Although these nodes are not broken in time unit ℓ and have may faithful links in this time unit, they may still be unable to recover from the break-in at the previous time unit. The reason is that they can only communicate with nodes that were also broken in the previous time unit, and thus they have no one that can actually help them to recover.

We therefore formulate the notion of an operational node in a recursive manner, to make sure that a node which is operational in time unit ℓ is able to communicate with nodes that were operational in time unit $\ell - 1$.

Definition 2 *Let n be the number of nodes in the network, let t be an integer $0 \le t \le n$, and let N be a node in the network. We say that N is* t-operational *during time unit ℓ ($\ell > 1$), if*
(a) *Node N is not broken in this time unit, and*
(b) *During the refreshment phase at the beginning of this time unit, node N had faithful links with at least $n - t + 1$ nodes which were t-operational at time unit $\ell - 1$.*

Initially (i.e., in time unit 1), the t-operational nodes are all the nodes which are not broken. (Note that as t grows, the requirement gets weaker, so 0-operational is stronger than 1-operational, etc.)

[1] For simplicity we assume that each node sends at most one message to each other node at each round.

[2] This time interval does not have to be a time unit. It can also be part of a time unit, or it can span several time units.

Definition 3 *A node is said to be t-disconnected in time unit ℓ, if it is neither broken nor t-operational in this time unit.*[3]

Finally, we can formally define the power of the adversary in terms of the above notions

Definition 4 *An adversary is* (r, s, t)-limited *if during every time unit,*
(a) *at most r nodes are broken, and*
(b) *at most s nodes are t-disconnected.*

Since in the sequel we view both broken and disconnected nodes as nodes which are impaired by the adversary, it would be more convenient for us to work with the following definition

Definition 4$'$ An adversary is (s, t)-limited if during every time unit, at most s nodes are either broken or t-disconnected.

2.4 Definition of Security

We are now ready to define our notions of security for the authenticated network compiler. We define the notions of resilience and awareness, where resilience refers to operational nodes, while Awareness refers to disconnected ones. For the following definition, let Λ be an authenticated network compiler, and assume that we apply Λ with security parameter[4] of 1^k.

Definition 5 **(1)** *The compiler Λ is* (s, t)-resilient *if for any application protocol Π and any polynomial-time (s, t)-limited adversary A, the probability that during the execution of $\Lambda\Pi_k$ a node which is t-operational is being impersonated is negligible in k.*
(2) *The compiler Λ is* (s, t)-aware *if for any application protocol Π and any polynomial-time (s, t)-limited adversary A, every t-disconnected node outputs a special alert signal (denoted $\perp$), except with probability which is negligible in k.*
(3) *The compiler Λ is* (s, t)-secure *if it is both (s, t)-resilient and (s, t)-aware.*

3 Background: Proactive Distributed Signature Schemes

In this section we define (somewhat informally) the notion of a PDS. Loosely speaking, a distributed signature scheme (DS) is a protocol by which a "global signing capability" is shared among a group of nodes. That is, any sufficiently large subset of these nodes are able to generate a signature on any message, but any smaller subset cannot generate any valid message-signature pair. Signatures can be verified using a global, publicly available verification key. A somewhat more detailed description follows.

A distributed signature scheme, resilient against some class of adversaries, consists of three parts: a key generation algorithm GEN, a distributed signing protocol SIG, and a (local) verification algorithm VER. The key generation algorithm GEN generates a sequence $\vec{s} = s_1, \cdots, s_n$ of shares of a signature key, together with a verification key v. Each node N_i out of n nodes is given s_i (and v) at the initialization of the system. Given an input message m to be signed, the nodes invoke protocol SIG to jointly generate a signature $\mathrm{SIG}\vec{s}(m)$, such that the verification algorithm with verification key v accepts the signature. That is, $\mathrm{VER}_v(m, \mathrm{SIG}_{\vec{s}}(m)) = 1$. Furthermore, any adversary of the considered class should be unable to generate a new message m' and a signature x such that $\mathrm{VER}_v(m', x) = 1$. See [13, Def. 1 and 2] and [16, Def. 3.1] for more complete definitions.

[3] Intuitively, a node is t-disconnected if there are at least t links that it cannot use in order to regain security during the refreshment phase.

[4] The security parameter determines the size of the cryptographic keys that are used by the compiler Λ. Although it is needed for the definition, we ignore this security parameter in the rest of this extended abstract.

In a proactive distributed signature scheme (PDS), the adversary cannot generate fake signatures (and, in particular, compute the signature key) as long as not too many nodes are broken into *at the same time unit.* For this purport, in addition to the protocols GEN, SIG, VER the nodes engage in each refreshment phase in a protocol REF for refreshing their shares of the signature key. We stress, however, that the global verification key must remain unchanged throughout. For a more formal definition of proactive distributed signature protocols, see [16, Def. 3.3]. In this extended abstract, we use the following informal definition-sketch:

Meta-Definition 6 *A PDS scheme* (GEN, SIG, VER, REF) *is said to be t-secure in a communication model with authenticated broadcast channels, if for any PPT adversary A which can break into at most t nodes in each time unit, the following properties hold*

1. *If at least $n - t$ unbroken nodes have a consistent view of the message m to be signed and of every message which is sent over the broadcast channels, then at the end of the protocol* SIG, *each of these nodes has a valid signature on m with respect to the verification key.*
2. *The adversary cannot obtain any pair (m, σ), where σ is a valid signature on m with respect to the fixed verification key, unless at least one node which is not broken participates in the signature protocol* SIG *on the message m.*

Several proactive distributed signature schemes were proposed recently [18, 16, 13, 12, 14]. In particular, the [14] scheme is t-secure for $t = \lfloor \frac{n-1}{2} \rfloor$. Also, in Section 7 we describe a simple scheme which has good on-line/off-line properties, but is only t-secure for $t = \lfloor \frac{n-1}{3} \rfloor$.

The (sketchy) security definition of a PDS scheme in our unauthenticated model is adapted from the above:

Meta-Definition 7 *A PDS scheme* (GEN, SIG, VER, REF) *is said to be (s, t)-secure in an unauthenticated communication model, if the following properties hold for any PPT (s, t)-limited adversary A:*

1. *If at least $n - s$ nodes which are t-operational have a consistent view of the message m to be signed, then at the end of the protocol each of these nodes has a valid signature on m with respect to the fixed verification key.*
2. *The adversary cannot obtain any pair (m, σ), where σ is a valid signature on m with respect to the fixed verification key, unless at least one t-operational node participates in the signature protocol* SIG *on the message m.*

4 Communicating Over Faulty Links

Our main objective in this chapter is to implement broadcast channels over our network of faulty links. This is used in Section 5 to modify PDS schemes which are designed for networks with broadcast channels into ones which can work in our model of unauthenticated links. To implement these broadcast channels we use tools for Byzantine agreement. Starting from [22, 19], there has been a large body of work regarding Byzantine agreement in various models. Some of these works, such as [7] and [11], are particularly useful to us. Yet, our model holds some additional complications, which force us to modify these protocols, and use some additional tools.

4.1 Achieving Connectivity

We start with a simple protocol which guarantees that the adversary must disrupt many links to prevent messages from being delivered. The protocol is just a 'two-phase echo'.

protocol DISPERSE
1. *To send a message m, node N_i sends m to all other nodes.*
2. *Each node forwards the message to all intended recipients.*
3. *Each node N_j checks all of the forwarded messages it got and accepts every message which is addressed to it (regardless of the number of copies it receives).*

We stress that the DISPERSE protocol does not guarantee that only authentic messages are accepted, nor does it ensure that messages are not re-transmitted. The only guarantee it offers is that if N_i, N_j are two nodes in the network, so that between N_i and N_j there is a path of length at most two consisting only of faithful links, then any message which is sent between these nodes using the DISPERSE protocol arrives at its destination. This motivates the following definition:

Definition 8 *A set S of nodes is called a* communication clique *during some time interval, if for any two nodes $N_i, N_j \in S$, there is a path of length at most two between N_i and N_j, consisting only of faithful links during this time.*

We observe that the definition of operational nodes implies that if there is any node N which is t-operational in time unit ℓ, then at the refreshment phase at the beginning of this time unit, there exists a communication clique of size at least $n - t + 1$ (namely, the set of nodes to which N is connected via faithful links).

4.2 Using Digital Signatures

Although the DISPERSE protocol offers some connectivity advantages, it does not offer any authenticity. In particular, the adversary can easily forge a message from N_i to N_j without breaking into any of them (and without even modifying any message on the direct link between them). To obtain some authenticity guarantees, we combine the DISPERSE protocol with digital signatures. We start by describing the framework in which we use these signatures.

Using certificates. We assume a network model in which all the nodes have access to a global verification key of some distributed signature scheme. This global verification key is initialized at system start-up, and it remains fixed throughout the lifetime of the system. (As was discussed in the Introduction, this model can be realized by storing the global key in ROM in each node.) In the protocols which we describe below, this global key is used to verify "certificates". Thus we denote it by $v_{\mathtt{cert}}$ in the sequel. To "actively participate" in the protocols below, a node must possess some additional cryptographic keys. To formalize this we have the following syntactic definition

Definition 9 *A node N_i is said to be* certificate-holding *in time unit ℓ, if during this time unit is has:*
(a) *A pair of public and secret keys for some 'standard' (non-distributed) signature scheme. We denote these keys by $s_i^{(\ell)}$ and $v_i^{(\ell)}$.*
(b) *A signature on the assertion "the public key of N_i in time unit ℓ is $v_i^{(\ell)}$"which is valid with respect to the global key $v_{\mathtt{cert}}$. We denote this certificate by $\mathtt{cert}_i^{(\ell)}$.*

(We stress again that the above definition is purely syntactic. In particular, it *does not* imply that the adversary "does not know" the secret-key $s_i^{(\ell)}$, or cannot generate certificates by itself.)

Certifying messages. A node N_i which is certificate-holding in time unit ℓ, may attempt to authenticate its communication in this time unit by attaching to a message m a signature σ, the public key $v_i^{(\ell)}$ which is needed to verify σ, and the certificate signature $\mathtt{cert}_i^{(\ell)}$. On the receiving end, a node N_j which receives a message $(m, \sigma, v, \mathtt{cert})$ from N_i, can check that σ is a valid signature on m w.r.t. v, and $\mathtt{cert}$ is a valid signature on the assertion "the public key of N_i in time unit ℓ is v" w.r.t. the global key $v_{\mathtt{cert}}$. Somewhat informally, we say that a message m is properly certified by node N_i in time unit ℓ if it has $(\sigma, v, \mathtt{cert})$ as above attached to it.

Authenticity guarantees. Even when using certificates as above, we still cannot always guarantee authenticity in our model (e.g., in the case of the impersonation attack described in the Introduction). Our strategy for obtaining authenticity in the implementation of PDS schemes in our model is as follows:

We start by defining some "favorable conditions" under which it is easy to obtain authenticity. In the next subsection we then present two protocols which obtain some 'broadcast properties' under these favorable conditions. In Section 5 we describe how we use these protocols to implement PDS over in model, and finally, in Section 6 we prove that these favorable conditions indeed hold (with high probability) whenever these protocols are used.

The "favorable conditions" which we need, in essence, mean that we have a communication clique in which the adversary does not forge messages. A little more formally, we have

Definition 10 *A set S of nodes in the network is an* authentication clique *during some time interval, if throughout this time interval*
(a) *S is a communication clique,*
(b) *All the nodes in S are certificate-holding, and*
(c) *For every two nodes $N_i, N_j \in S$, every message that N_j receives from N_i and is successfully verified, indeed originated with N_i in this time interval.*

It is easy to see that using certificates in conjunction with the DISPERSE protocol, results in virtual "reliable links" between all the nodes in an authentication clique. Namely,

Observation 11 *Assume that S is an authentication clique during some time interval, and that the nodes in S properly certify all their communication and use the* DISPERSE *protocol to send these certified messages. Then, for every two nodes $N_i, N_j \in S$, the properly certified messages which N_j receives from N_i are all and only the ones which N_i sent during this time interval.*

4.3 Broadcast Channels

Roughly, broadcast channels guarantee the following additional property, on top of point-to point communication: If a party receives a message, then all parties receive the same message. We show how we emulate broadcast-like channels in our model of an unauthenticated network. In the sequel, there are two different scenarios in which we need to use these broadcast channels. One scenario is where the sender is certificate-holding. Here we can achieve full-fledged broadcast using known primitives. In the other scenario the sender is not certificate-holding. In this case we will need (and achieve) only a weaker broadcast-like property.

Implementing Strong Broadcast. We start with the case where the node that attempts to broadcast a message is certificate-holding. In this case we require that if there exists an authentication clique in the network, then within this authentication clique we emulate a full-fledged broadcast channel (even if the sending node deviates from the protocol).

For this we can use standard Byzantine agreement protocols from the literature. If our goal is to withstand $\lfloor \frac{n-1}{2} \rfloor$ broken nodes, then we can use the protocol of Dolev and Strong [7], which works in a model where digital signatures are available. In our model, the parties must use DISPERSE to send each message in the protocol. This protocol works in our model since the sender is able to sign its message, and within the authentication clique there are virtual "reliable links" and digital signature are available. Alternatively, if we can assume an authentication clique of size $\lfloor \frac{2n-1}{3} \rfloor$ broken nodes, we can use the protocol of Feldman and Micali [11], which runs in constant expected number of rounds. In any case, if we have a large enough authentication clique, then by using these protocols we guarantee that all the nodes in the clique agree on the same value.

Given a Byzantine agreement protocol, the broadcast protocol is simple: the sender first sends its (signed) message to all parties. Next the parties reach agreement on whether the massage was received. Note that since the message is signed, there is no ambiguity about the contents of this message in case it was received.

Implementing Weak Broadcast

We now turn our attention to the case where the node which attempts to send a message on the broadcast channel is not certificate-holding. We note that in this case, using the Dolev-Strong Byzantine agreement protocol is more problematic, since the sender is unable to sign its original message. However, it turns out that in this case we only need a much weaker broadcast property. Informally, we need to make sure that at the end of the protocol there is a single value so that every node either receive this value, or do not receive any value at all. We call this channel a "partial broadcast channel". Indeed, this channel can be emulated using a more efficient protocol than a general-purpose Byzantine agreement protocol.

We start by describing a protocol for obtaining "partial agreement". In this protocol, each node starts with some input value, and at the end of the protocol it outputs either an output value or the special symbol ϕ (which means "no output"). Since the participants in this protocol have to properly certify their messages, then only nodes which are certificate-holding can participate in it.

Lemma 12 *If during an execution of* PARTIAL-AGREEMENT*, there exists an authentication clique S of size at least $\lceil \frac{n+1}{2} \rceil$, then in this execution,*
1. *If all the nodes in S start with the same input value x, then they all output x as the output value.*
2. *If one of the nodes in S outputs a value $x \neq \phi$, then all the other nodes in S output either x or ϕ.*

Proof: **1.** Since S is an authentication clique then between any two nodes $N_i, N_j \in S$, the messages which N_j receives from N_i are all and only the messages which N_i sends. Moreover, since each of these nodes only certifies a single input value x, then no node in S ever marks any other node in S "cheater".

Since they all send the same input value x, then this value will be a majority value in all of them in Step 2, and moreover, in each $N_i \in S$ we have $S \subseteq \textsc{maj}_i$. This is still true in Step 3, and so $|\textsc{maj}'_i| \geq |S| \geq \lceil \frac{n+1}{2} \rceil$ for all $N_i \in S$, hence all the nodes in S will output the value x.

<u>protocol PARTIAL-AGREEMENT</u>

1. *Node N_i, which has input value x_i, properly certifies x_i and uses* DISPERSE *to send it to the other nodes.*

   (We note that nodes which are not certificate-holding cannot send messages in this phase, but they still participate in forwarding messages in the DISPERSE *protocol.)*

2. *Node N_i collects all the properly certified messages which it received in Step 1. Every node for which N_i receives properly certified messages on more than one input value is marked "cheater".*

   If there exists a set $\textsc{maj}_i$ of nodes which are not marked "cheaters", so that $|\textsc{maj}_i| \geq \lceil \frac{n+1}{2} \rceil$ and all the nodes in $\textsc{maj}_i$ have the same input value, then we denote this value by y_i. If there is no such set then we denote $y_i = \phi$, and $\textsc{maj}_i = \emptyset$.

   Node N_i uses DISPERSE *to send again all the properly certified messages from all the nodes in $\textsc{maj}_i$.*

3. *Each node N_i collects all the properly certified messages which it received in Steps 1 and 2. Every node for which N_i receives properly certified messages on more than one input value is again marked "cheater".*

   Denote by $\textsc{maj}'_i$ the set of all the nodes which were in $\textsc{maj}_i$ in Step 2, and were not marked "cheaters" in Step 3. If $\textsc{maj}'_i$ still contains at least $\lceil \frac{n+1}{2} \rceil$ nodes, then y_i is the output value of N_i. Otherwise, N_i outputs ϕ.

2. Assume to the contrary that $N_i \in S$ outputs $y_i \neq \phi$, and $N_j \in S$ outputs $y_j \neq y_i$ (and also $y_j \neq \phi$). Since both $\textsc{maj}'_i$ and $\textsc{maj}'_j$ contain at least $\lceil \frac{n+1}{2} \rceil$ nodes, then there exists at least one node $N_k \in \textsc{maj}'_i \cap \textsc{maj}'_j$.

However, N_k appears in $\textsc{maj}'_i$ with input value y_i and in $\textsc{maj}'_j$ with input value y_j. Therefore, in Step 2 N_i and N_j send (using DISPERSE) two different messages, both certified by N_k, where one message states that the input value of N_k is y_i and the other states that the input of N_k is y_j. Since S is an authentication clique then these messages reach both N_j and N_i, thus N_k is marked "cheater" by both of them. Contradiction. □

In the sequel, we use the PARTIAL-AGREEMENT protocol to obtain partial broadcast as follows: A node N which is not certificate-holding and wants to send a message over a partial broadcast channel, simply sends its message "in the clear" to all the other nodes. All the certificate-holding nodes now execute the PARTIAL-AGREEMENT protocol on the messages which they received from N. Lemma 12 implies that if there exists an authentication clique in the network, then within this authentication clique the procedure above emulates a partial broadcast channel.

5 Proactive Signatures over Unauthenticated Channels

In this section we show how *any PDS scheme* which is t-secure in the model with broadcast channel can be modified to an (t, t)-secure PDS scheme in our model, for the same value of t.

For the rest of this section let (GEN, SIG, VER, REF) be a PDS scheme which is t-secure in a communication model with authenticated broadcast channels (for some $t \leq \lfloor \frac{n-1}{2} \rfloor$). In addition, we use a 'standard' (non-distributed) signature scheme (GEN$'$, SIG$'$, VER$'$) which is secure against existential forgery in the sense of of [15].

The main idea is to maintain (in an inductive manner) the following invariant throughout the lifetime of the system:

Invariant: In every time unit, there exists an authentication clique S of size at least $n - t$, where all the nodes in S are t-operational and they all have valid shares of the signature key of the PDS (so they are able to sign messages).

We now give a detailed description of our construction. We postpone the analysis to Section 6.

5.1 System Set-up

In the system set-up, the algorithm GEN is executed to generate the fixed global verification key v_{cert} and the corresponding sequence of signature key shares $s_1, \ldots, s_n$. The verification key is then burned in ROM together with the protocol code, and installed in each of the nodes. Each node N_i is also given the share s_i of the corresponding signature-key of the PDS scheme. This share *is stored in regular memory*.

In addition to these global keys, each node N_i is also given a pair $(s_i^{(1)}, v_i^{(1)})$ of secret- and public-keys (which are generated by the key-generation algorithm GEN' of the standard signature scheme) and a valid signature (with respect to the global key v_{cert}) on the message "the public key of N_i in time unit 1 is $v_i^{(1)}$".[5]

5.2 Signing a Message

To sign a message m during time unit ℓ, the nodes follow the signature protocol SIG of the original PDS, with the following modifications:

1. Nodes which are not certificate-holding in this time unit do not participate in the protocol, (except for forwarding messages during the DISPERSE protocol).
2. Every message which is sent in SIG on a broadcast channel, is sent in our implementation using the protocol for strong broadcast from Section 4.

We note that as long as the above invariant holds, then by Lemma 12, the nodes in the authentication clique have a consistent view of all the messages which are sent on the broadcast channels during the execution of SIG. By the first property in Definition 6, these nodes then obtain a signature on m at the end of the protocol.

5.3 Refreshment Phases

The refreshment phases are divided into two parts: In the first part the nodes generate new personal keys, and in the second part they execute the refreshment protocol REF of the original PDS scheme.

Part 1. At the beginning of time unit ℓ, each node N_i picks a new pair $(s_i^{(\ell)}, v_i^{(\ell)})$ of personal signature and verification keys (by running the algorithm GEN'). Next, N_i sends its new personal verification $v_i^{(\ell)}$ key to all the other nodes.

(This message is sent "in the clear", since N_i may be trying to recover from a break-in, and thus it may not have the necessary keys in order to authenticate its communication. Notice, however, that if N_i is to be operational in this time unit, then it must have many faithful links, in which case many nodes receive the correct value of $v_i^{(\ell)}$.)

[5] Below we refer to $(s_i^{(\ell)}, v_i^{(\ell)})$ as the personal keys of node N_i. Since the nodes have to pick new personal keys at each refreshment phase, it is preferable to use a signature scheme where choosing a new pair of keys does not require much work. In particular one can use an El-Gamal based scheme, where choosing new keys requires only one modular exponentiation.

The nodes now invoke the PARTIAL-AGREEMENT protocol using the (unauthenticated) message from N_i as their input. Then, they invoke the modified signature protocol from above, using the output of PARTIAL-AGREEMENT as a message to be signed. Namely, a node N_j which has output $y_j \neq \phi$ from PARTIAL-AGREEMENT, participates in the protocol SIG to obtain a signature on the message "the public key of N_i in time unit ℓ is y_j". If N_j obtains a signature on that message, then it sends this signature to N_i.

Part 2. Once valid certificates are obtained, the nodes execute the share refreshment protocol REF of the PDS scheme itself, in order to refresh their shares of the global signature key. The REF protocol is modified in the same manner as the SIG protocol. Here nodes can use either their new certificates or certificates from the about-to-end time unit to participate in this step.

Theorem 13 *If* $(\text{GEN}, \text{SIG}, \text{VER}, \text{REF})$ *is a PDS scheme which is t-secure in the model with broadcast channel, then the resulting scheme is (t,t)-secure in the model with unauthenticated links.*

6 A Compiler for Unauthenticated Networks

Using the PDS scheme from Section 5, we can now describe our construction of a compiler for unauthenticated networks. Let Π be an application protocol which is designed to work over authenticated networks. The compiled protocol, denoted $\Lambda_{cert}\Pi$, proceeds as follows:

System set-up. The nodes initialize a PDS scheme, as described in Subsection 5.1.

Refreshment phases. The nodes execute the refreshment protocol as described in Subsection 5.3. A node who fails to obtain a certificate for its newly chosen personal key or to refresh its share of the global signature key outputs an `alert` signal.

Security transformations. The certificates which were obtained in the refreshment phase are used to authenticate the application-protocol messages, as follows. At the sending node, each Π-message is properly certified using the current personal signature key and the current certificate. The receiving node verifies the certificate using the global key in its ROM, and then verifies the signature using the attached verification key. If any verification fails then the receiver discards the message (and may also output some sort of a `warning` signal).

Alternatively, each pair of communicating nodes can use the certified keys to exchange symmetric session keys (using standard techniques) and use those to authenticate their communication.

Theorem 14 *Let* $(\text{GEN}, \text{SIG}, \text{VER}, \text{REF})$ *be a PDS scheme for n nodes which is t-secure in a model with an authenticated broadcast channel (in the sense of Definition 6) for some $t \leq \lfloor \frac{n-1}{2} \rfloor$. Also let* $(\text{GEN}', \text{SIG}', \text{VER}')$ *be a standard (non-distributed) signature scheme which is existentially unforgeable in the sense of [15].*

Then the compiler Λ_{cert} above using these two signature schemes, is (t,t)-secure (in the sense of Definition 5) for the same value oft.

We remark that PDS schemes can be constructed, using standard general secure multiparty protocols, from any existentially unforgeable signature scheme. Schemes that are far more efficient exist based on the El-Gamal, DSS, and RSA signature schemes [18, 16, 13, 12, 14].

Proof (sketch): Let Π be any application protocol for n nodes, and let $(\text{GEN}, \text{SIG}, \text{VER}, \text{REF})$ be a PDS scheme which is t-secure in a communication model with authenticated broadcast channels (for some $t \leq \lfloor \frac{n-1}{2} \rfloor$). Denote the protocol which results from compiling Π using this PDS scheme by $\Lambda_{cert}\Pi$. The main lemma which we need in this proof is

Lemma 15 *Let A be any (PPT) adversary which is (t,t)-limited. Then the following three conditions hold in every time unit during the execution of $\Lambda_{cert}\Pi$, except with negligible probability.*

1. *Every node N_i which is t-operational in time unit ℓ obtains a certificate for its personal signature key and a refreshed share of the global signature key in the refreshment phase at the beginning of this time unit.*

2. *At each time unit ℓ, and for each node N_i, at most one signature on a certificate of the form* "the public key of N_i in time unit ℓ is v" *is generated in the refreshment phase at the beginning of this time unit.*

3. *If N_i obtains a certificate in the refreshment phase at the beginning of time unit ℓ for its personal public key $v_i^{(\ell)}$, and if N_i is not broken during this time unit, then the only properly certified messages which other nodes obtain from N_i during time unit ℓ are the ones which N_i sends in this time unit.*

Proof (sketch): We first observe that Part 2 of the above claim implies Part 3, since if N_i obtained a certificate for time unit ℓ, then the only certified verification key for N_i in this time unit is the one chosen by N_i itself. Thus the adversary does not know this key and cannot forge messages with respect to it (except with negligible probability).

It is therefore sufficient to prove only Parts 1 and 2, and we prove them by induction on the time units ℓ. Assume that the lemma holds until time unit $\ell - 1$, and we prove for time unit ℓ. Let N_i be t-operational in time unit ℓ. By definition, this implies that during the refreshment phase at the beginning of this time unit, N_i has faithful links to at least $n - t + 1$ nodes which were t-operational in time unit $\ell - 1$ (and it follows the protocol). Denote this set of nodes by S. Thus, S is a communication clique during this refreshment phase, and by Parts 1 and 3 of the induction hypothesis, S is also an authentication clique.

Since N_i has faithful links to all the nodes in S, then they all receive the verification key $v_i^{(\ell)}$ which is sent by N_i at the beginning of this refreshment phase. Thus, by Part 1 of Lemma 12, at the end of the PARTIAL-AGREEMENT protocol they all agree to participate in signing $v_i^{(\ell)}$.

Since the nodes in S form an authentication clique and since we use strong broadcast protocol to send messages during the execution of SIG, then the nodes in S have a consistent view of every message which is sent on the broadcast channels, and thus at the end of the signature protocol they all have a valid signature on the certificate of N_i. Then they all send this certificate to N_i, and since they have faithful links to N_i it receives this certificate.

A similar argument shows that the share-refreshment protocol is also be successful for N_i, thus concluding the proof of Part 1.

To prove Part 2, we note that since the adversary is (t,t)-limited, then there must exist a node N which is t-operational in time unit ℓ (this node may be different than N_i, however). As we explained above, this implies that during the refreshment phase at the beginning of time unit ℓ there exists an authentication clique S of size $\geq (n - t)$. Since $|S| \geq n - t$ then a signature on any value requires the cooperation of at least one node in S.

Assume, then, that a certificate of the form "the public key of N_i in time unit ℓ is v" is generated in this refreshment phase, and let F be a node in S which participates in generating the signature on this certificate. But since the nodes in S execute a PARTIAL-AGREEMENT to agree on the value to be signed, then Part 2 of Lemma 12 implies that no node in S participates in the signature on any other certificate for node N_i in this time unit, and thus no other certificate can be signed (except with negligible probability). □

Lemma 15 directly implies Theorem 13. □

7 A proactive distributed signature scheme

Motivation. As described in Sections 5 and 6, we can transform any PDS schemes which is secure in a model with broadcast channel to one which can be used over our unauthenticated channels, and then use this scheme in the construction of the compiler Λ_{cert}. One drawback of many of the known distributed signature protocols, is that they tend to have high round and communication complexities. Moreover, since we use the DISPERSE and Byzantine-agreement protocols to implement each broadcast, then the communication complexity gets even much worse. Below we describe a distributed signature protocol which may be more adequate for our needs.

The main advantage of this scheme is that almost all the work can be done off-line (i.e., independently of the message to be signed), where the on-line part (i.e., the part which depends on the message) is minimal, and in particular non-interactive. (Off-line vs. on-line work was first studied in the context of signature schemes in [8].) In our construction from Section 5, the off-line part of the signing process on a new personal verification key can be conveniently carried out *at any time during the previous time period*, prior to the current refreshment phase. Only the on-line part will be carried out at the beginning of a phase. Therefore, the refreshment phases are kept short, and the bulk of the work can be done off-line.

On the down side, this scheme is only t-secure for $t = \lfloor \frac{n-1}{3} \rfloor$, as opposed to $t = \lfloor \frac{n-1}{2} \rfloor$ in some of the other schemes.

7.1 Overview

The scheme which we present here is based on a variant of El-Gamal signatures [9]. Herzberg et al. describe in [16] a transformation which can be applied to any El-Gamal based distributed signature scheme (DS) in order to achieve a PDS scheme which is secure in model with broadcast channel. Thus, below we only describe the distributed protocol itself, without the proactive part.

In this scheme (like many others), each node's share of the signature key is used to generate a "signature part" for the given message. Then the nodes collaborate to combined all these "signature parts" into one valid signature. The main difference between this signature scheme and previous ones is in the way we combine the signature parts into a valid signature, and in particular with the way we handle the signature-parts of faulty nodes (which may be corrupted).

In other schemes (e.g., [18, 17, 16]) the correctness of each individual signature-part is verified before inclusion in the global computation. Instead, in the scheme below we make use of error-correcting codes to correct erroneous or missing signature parts. (This technique draws from [5].) That is, correct signature parts form a codeword, so each node can use error correction algorithms to correct the "bad signature-parts" (as long as there aren't too many of them), before combining all them into a valid signature.

El-Gamal Signatures. In any El-Gamal signature, the verification key is a triple (p, g, y), where p is a large prime, g is a generator in Z_p^* and y is another element in Z_p^*. The corresponding signature key is $x \in Z_p^*$ so that $g^x = y \pmod{p}$. In the variant underlying our scheme, the signature on a message m with respect to a public key (p, g, y) is a pair (u, w) so that $y^{H(m,u)} = g^w \cdot u^u \pmod{p}$ (where H is an appropriate hash function). A signer who knows x (the discrete-log of y) can generate this signature by picking a random element $k \in Z_p^*$ and setting $u \leftarrow g^k \ (mod\ p)$ and $w \leftarrow x \cdot H(m,u) - k \cdot u \pmod{p-1}$.

We partition the signing protocol to an off-line and on-line stages. In the off-line stage the nodes share the secret x and the random number k (so each node N_i has shares x_i, k_i), and compute

$u = g^k \pmod p$. In the on-line stage the message m is sent to all the nodes, and each node computes its own signature-part w_i and sends (u, w_i) to the verifier. The verifier then applies an error-correction procedure to the w_i's to "weed-out" the bad parts and combines everything into one valid signature.

Sharing k and x. To share k, x the nodes use Feldman's t-out-of-n Verifiable-Secret-Sharing procedure [10]. At the end of this protocol both the secret x and the random value k are shared among the nodes in the following manner: There exists two t-degree polynomials $X(\cdot)$ and $K(\cdot)$ so that

- $x = X(0) \pmod{p-1}$ and $k = K(0) \pmod{p-1}$; and
- Each node N_i has shares $x_i = X(i) \pmod{p-1}$ and $k_i = K(i) \pmod{p-1}$.

An artifact of using Feldman's procedure is that at the end of the protocol each node also knows the value $u = g^k \pmod p$. This protocol uses in a crucial way the authenticated broadcast channel to ensure that all the "honest" nodes have a consistent view.

Computing signature parts. Given a message m, each node N_i (who knows k_i, x_i and u) sets $w_i \leftarrow x_i \cdot H(m,u) - k_i \cdot u \pmod{p-1}$. The signature-part of node N_i is (u, w_i).

Notice that since m, u are the same for all nodes (and hence, so is the value $H(m,u)$), then there exists a t-degree polynomial $W(\cdot) = H(m,u)X(\cdot) - uK(\cdot)$ so that $w_i = W(i) \pmod{p-1}$ for all i. Thus the vector of w_i's form a codeword over a Generalized Reed-Solomon code.

Combining the signature parts. Upon receiving all the signature-parts, the verifier first finds the majority value for u. Then it takes the vector of w_i's (some of which may be corrupted), and applies to it the Berlekamp-Welch decoding procedure [3]. This procedure is guaranteed to compute the coefficients of the polynomial $W(\cdot)$ as long as no more than $(n-t)/2$ of the w_i's are corrupted. After finding $W(\cdot)$, the verifier computes $w \leftarrow W(0) \pmod{p-1}$. Notice that (u, w) is a valid signature on m with respect to p, g, y, since

$$\begin{aligned} w &= W(0) \\ &= H(m,u)X(0) - uK(0) = x \cdot H(m,u) - k \cdot u \pmod{p-1} \end{aligned}$$

and thus $y^{H(m,u)} = g^{x \cdot H(m,u)} = g^{w + k \cdot u} = g^w u^u \pmod p$.

Remark 1. One drawback of this scheme is that the Berlekamp-Welch decoder can correct at most $(n-t)/2$ corrupted parts, so this scheme is only $(\lfloor \frac{n-1}{3} \rfloor)$-secure.

Acknowledgments We wish to thank Mihir Bellare, Juan Garay, Oded Goldreich, Hugo Krawczyk, Tal Rabin and Moti Yung for very helpful discussions and comments.

References

[1] Mihir Bellare and Phil Rogaway. Entity authentication and key distribution. *Proceedings of CRYPTO'93,* pages 232–249, August 1993.

[2] Mihir Bellare and Phil Rogaway. Provably secure session key distribution — the three node case. in *proceedings of STOC'95*, pages 57–66.

[3] E. Berlekamp and L. Welch Error Correction of Algebraic Block Codes. US Patent Number 4,633,470.

[4] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993. Special issue on Secure Communications. (Preliminary version in *Crypto 91.*)

[5] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. *proceedings of 20th STOC,* pp. 1-10, 1988.

[6] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Proceedings of CRYPTO'94*.

[7] D. Dolev and H. R. Strong, Authenticated Algorithms for Byzantine Agreement. SIAM J. of Computing, 12(4), pp. 656-666, 1983

[8] S. Even, O. Goldreich and S. Micali. On-line/off-line digital signatures. In *Journal of Cryptology*, vol. 9, No. 1, 1996, Pages 35–67. Springer-Verlag.

[9] T. El-Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *IEEE Trans. Information Theory*, vol. 31, 1985, pp. 469-472

[10] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In proceedings of *28th FOCS*, IEEE, 1987.

[11] P. Feldman and S. Micali. Optimal Algorithms for Byzantine Agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pp. 148-161, 1988.

[12] Y. Frankel, P. Gemmell, P. Mackenzie and M. Yung, Proactive RSA. In Proceedings of CRYPTO'97.

[13] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Robust Threshold DSS Signatures. in *Proceedings of EUROCRYPT 96*.

[14] R. Gennaro, T. Rabin and M. Rabin, "VSS and Secure Multiparty Computations Made Simple", manuscript.

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

[16] A. Herzberg, M. Jakobson, S. Jarecki, H. Krawczyk and M. Yung. Proactive public key and signature systems. *Proceedings of 4th ACM Conference on Computer & Communications Security*, to appear.

[17] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung. Proactive Secret Sharing or: How to Cope with Perpetual Leakage. In *Proceedings of CRYPTO'95*.

[18] S. Jarecki. Proactive Secret Sharing and Public Key Cryptosystems. M.Eng. thesis, MIT, 1996

[19] L. Lamport, R. Shostak and M. Pease. The Byzantine generals problems. *ACM Trans. on programming languages and systems*, 4(3), pp. 382-401, 1982.

[20] N. Lynch. Distributed Algorithms. Morgan Kaufmann Publishers Inc. 1996.

[21] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the* 10^{th} *Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada*, pages 51–59, 1991.

[22] M. Pease, R. Shostak and L. Lamport. Reaching agreements in the precence of faults. In *Journal of the ACM,* 27(2), pp. 228-234, 1980