J. Cryptology (2000) 13: 61–105
DOI: 10.1007/s001459910004

Journal of
CRYPTOLOGY

Maintaining Authenticated Communication in the Presence of Break-Ins*

Ran Canetti
IBM T.J. Watson Research, PO Box 704,
Yorktown Heights, NY 10598, U.S.A.
canetti@watson.ibm.com

Shai Halevi
IBM T.J. Watson Research, PO Box 704,
Yorktown Heights, NY 10598, U.S.A.
shaih@watson.ibm.com

Amir Herzberg
IBM Haifa Research Lab - Tel Aviv Annex 2,
Weizmann Institute, Tel Aviv, Israel
amirh@vnet.ibm.com

Communicated by Oded Goldreich

Received 22 April 1998 and revised 2 April 1999

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 transient periods of time. Once a party is broken into, its cryptographic keys are exposed and perhaps modified. Yet, when aided by other parties it should be able to *regain* its ability to communicate in an authenticated way. We present a mathematical model for this highly adversarial setting, exhibiting salient properties and parameters, and then describe a practically appealing protocol for solving this problem.

A key element in our solution is devising a *proactive distributed signature* (*PDS*) *scheme* in our model. The PDS schemes known in the literature are designed for a model where authenticated communication is available. We therefore show how these schemes can be modified to work in our model, where no such primitives are available a priori. In the process of devising these schemes, we also present a new definition of PDS schemes (and of distributed signature schemes in general). This definition may be of independent interest.

Key words. Authentication protocols, Break-ins, Recovery, Distributed signatures, Proactive signatures, Proactive protocols.

* An extended abstract of this work appeared in the *Proceedings of Principles of Distributed Computing*, 1997. Part of this research was done while Ran Canetti and Shai Halevi were at the Laboratory of Computer Science in MIT.

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 usually easier to achieve.

Authenticated communication over adversarially controlled links can be achieved using a variety of standard cryptographic techniques, provided that some cryptographic keys are distributed ahead of time (see, for instance, [15], [5], [25], [8], [9], and [2]). 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 communication systems are often based on obtaining (or even modifying) these keys through system penetration, rather than via 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.

These questions are at the heart of the proactive approach to cryptography [30], [13]. This approach is aimed at maintaining security of cryptosystems in the presence of repeated break-ins. As in the "threshold cryptography" approach [14], the proactive approach calls for distributing the cryptographic capabilities (e.g., the signing key of a signature scheme) among several servers, and designing protocols for the servers to carry out the task at hand (e.g., generating signatures) securely as long as not too many servers are broken into. In addition, a proactive solution introduces periodic refreshment phases where the servers help each other to regain their security from possible break-ins. Consequently, a proactive system remains secure as long as not too many servers are broken into *between two invocations of the refreshment protocol*. See [19] for a survey on threshold cryptography and [12] for a survey on proactive security.

Contributions of this paper. This paper can be viewed in two levels: On a conceptual level, we investigate the problem of proactive authenticated communication in the presence of repeated break-in attacks, devise a framework for analyzing it, and describe a protocol for solving the problem within this framework. On a more technical level, our main contribution is a transformation, converting any secure Proactive Distributed Signature (PDS) scheme that works over reliable communication links into one that can work over faulty (i.e., adversarially controlled) links.

We begin by defining a formal model for the problem of proactive authentication over faulty links, and exhibiting some fundamental properties and parameters of this model. Next we concentrate on the construction of secure PDS schemes in this model. We present a definition of secure PDS schemes, and show how these can be constructed in our model. (This definition provides an alternative approach to that of Gennaro et al. [20].) Finally we describe and analyze a general method for transforming any protocol that works over reliable communication links into one that can work over faulty links. This method uses in an essential way the PDS schemes constructed before. We stress that proactive authentication is essential for realizing any of the known proactive protocols (e.g., [30], [13], [24], [23], [18], and [32]) in a realistic system.

1.1. *The Model of Computation*

We consider a synchronous network of nodes, which communicate via "totally unreliable" communication links and are also prone to break-ins. That is, an 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 compromise nodes occasionally, thus obtaining all their secret data; even further, this secret data may be modified by the adversary. In this highly adversarial model, we address the problem of recovering from break-ins and regaining authenticated communication. To make a solution possible we limit the adversary in the following three ways.

First, if the adversary can modify the code of the recovery protocol itself, then 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 and its realization in Section 6.)

Second, it is not hard to see that no protocol can guarantee authenticated communication if the adversary can simultaneously break into all the nodes in the network, before the nodes had a chance to proactively "refresh" their internal data. We thus assume some bound on the number of nodes that are broken into between any two successive proactive refreshment phases. Below we refer to the time between two successive refreshment phases as a time unit. We consider adversaries that only break into a minority of the nodes in any time unit.

Third, a solution may still be impossible even if only a few nodes are broken into at any given time, since the adversary can block the messages needed for recovery. To overcome this, we further limit the number and connectivity of links on which the adversary injects, corrupts, or deletes messages.

To gain some intuition into the problems involved in maintaining authenticated communication in our model, it is instructive to consider the following attack: 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. Even worse, if the adversary modified the cryptographic keys within N's memory, then N cannot verify incoming messages, and so the adversary can also impersonate the other nodes in the eyes of N.

Jumping ahead, our approach toward dealing with this seemingly hopeless situation, *when a node cannot even trust its own memory*, proceeds as follows. We allow each node to maintain a small piece of data in ROM, together with its program. This piece of data is chosen once at system set-up and remains public and unmodified throughout the life of the system. It holds a verification key of a PDS scheme, whose signing capabilities is shared by the nodes. This way, as long as the PDS scheme remains secure, a node will be able to be *aware* of the fact that it is being impersonated. In addition, we present a protocol which guarantees that the adversary must "cut off" N from a large fraction of the network in order to impersonate it, and hence a reasonably restricted adversary cannot simultaneously impersonate too many nodes.

1.2. *Defining our Goals*

Formulating a reasonable notion of solution to the proactive authentication problem is an important contribution of this paper. In general, ensuring secure communication over faulty links involves two requirements:

- *Authenticity.* Nodes do not accept as authentic incoming messages that were modified (or injected) by the adversary.
- *Delivery.* Messages that are sent over the network arrive at their destinations.

In most settings, these requirements can be dealt with separately. Indeed, most of the authentication mechanisms in the literature (e.g., [4], [5], and [2]) only deal with ensuring authenticity and explicitly putting aside the delivery issue. The proactive setting is different, however, since recovering from break-ins requires that nodes be able to communicate with each other. Hence in this work we are forced to combine the authenticity and delivery issues, and so we construct authentication mechanisms that also guarantee delivery (provided the adversary is reasonably restricted). In what follows we refer to links that enjoy both authenticity and delivery as reliable links.

Authenticators. Our solution to the proactive authentication problem is centered around the notion of a proactive authenticator. Informally, an authenticator is a *compiler* that transforms a protocol that expects reliable links into one that can also work over faulty links.

In defining this notion we follow the general paradigm used for defining secure multiparty protocols [28], [1], [10]: We first precisely define the "real-life" model of computation. We call this the unauthenticated-links (UL) model. Next we formalize the "idealized" model of computation that we want to emulate: here the adversary has similar capabilities, with the exception that it must deliver messages faithfully on the links and it cannot inject new messages. Call this the authenticated-links (AL) model. The notion of "emulation" is formulated as follows: we define the global output of a protocol, aimed at capturing the "functionality" of the protocol; next we say that a protocol π' in the UL model emulates a protocol π (designed for the AL model) if, for any (reasonably limited) adversary $\mathcal{U}$ in the UL model, there exists an adversary $\mathcal{A}$ in the AL model such that the global output of running π with $\mathcal{A}$ is indistinguishable from the global output of running π' with $\mathcal{U}$. Finally we define a proactive authenticator Λ as a compiler with the property that, for any protocol π, the protocol $\pi' = \Lambda(\pi)$ emulates π in the UL model.

Awareness. Although emulation guarantees that π' and π have similar functionality from a global point of view, it still allows the adversary to impersonate a node N occasionally. (This is inevitable, say if N is being "cut off" the network as described above). We thus complement the definition of a proactive authenticator by requiring that an impersonated node be aware of its situation. That is, we require that whenever a node is being impersonated, it outputs a special `alert` signal, which (in most realistic settings) can then be handled by a higher layer protocol or an operator. In addition, we consider a weaker form of awareness (which can potentially be achieved against stronger adversaries), where the `alert` signal is guaranteed only at the first time unit where impersonations occur (see Section 5.1).

Awareness is a somewhat nonstandard aspect of our treatment, since it represents a security requirement on the behavior of "nonfunctional" nodes. However, we view it as a crucial aspect for any "real world" solution and invest much effort in achieving it. For one thing, in the practice of computer security, the system managers and security officers are well equipped to regain control and expose the attackers, once an attack is detected. Moreover, awareness is important in complementing our model assumptions: in order to obtain a solution we need to assume that the adversary not only cannot break into too many nodes at the same time, but it also cannot modify or inject messages on too many links at the same time. In "real life," however, injecting messages on links is often much easier for an attacker than modifying existing messages, and the latter is often easier than breaking into nodes. (For example, injecting messages on the Internet is almost trivial: all you need to do is to construct IP packets with a modified source address. Deleting or modifying messages is harder, as it usually requires that the adversary attacks the routing mechanism as well.) Viewed in this light, awareness ensures that the adversary gets "worse results" by mounting the "easy attacks," since in this case its actions will be detected.

Related definitions. The authentication problem has been considered in a large number of works (e.g., [4], [7], [5], [15], [25], [8], [9], and [2]). The notion of authenticators was also adopted by Bellare et al. [2]. Yet, the model in [2] differs from the one here in several important aspects. First, they do not deal with recovery from break-ins: in their model, once a node is "corrupted" it remains so throughout. This frees them from having to provide *reliable* communication, and thus they can afford to not assume any limit on the number of links or nodes that the adversary may tamper with or corrupt. Next, they deal with *asynchronous* networks, where here synchrony is necessary in order to allow for joint refreshment phases. Finally, they emphasize the notion of independent *sessions* run by the same parties. (We remark that, although we do not deal with sessions in this work, they can be incorporated here in a similar way as there.)

1.3. *Our Solution*

We take the following approach. At the beginning of each refreshment phase every node chooses at random a pair of signing and verification keys of some (nondistributed) signature scheme. These keys are meant to be used for authentication in a standard way. Namely, a node signs each outgoing message, and a received message is accepted only if verification of the senders signature succeeds. (Alternatively, nodes can exchange symmetric session keys and use them to authenticate messages.)

The crux of the problem is how to get the newly chosen verification key to the other nodes. Simply sending the verification key (perhaps signed using the old signing key) will not do: consider a node N that is just recovering from a break-in. N's old signing key is compromised. Thus, the adversary can successfully impersonate N by forging N's signature and sending a fake new verification key in the name of N. Furthermore, N will not be aware of this impersonation. 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 signing key, so N can verify that M got its new public key. However, this may not work, since the adversary can erase M's verification

key in N's memory while N is broken into. Furthermore, it can replace this verification key by a fake one, for which the adversary knows the corresponding signing key. 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.

The main technical tool that we use to counter such attacks is a proactive distributed signature (PDS) scheme. Distributed signature schemes can be very roughly described as follows: As in any other signature scheme, there is a public key, which is used to verify signatures. The corresponding secret key, however, is not kept by any single node but is shared among the nodes in a way that enables any large enough subset of them jointly to sign a given message. At the same time, the adversary cannot forge signatures, even after breaking into some of the nodes. In a proactive distributed signature scheme, the nodes refresh their shares of the private signing key at each refreshment phase. This is done in a way that prevents the adversary from forging 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 signing key are changed periodically, the public verification key *remains unchanged throughout*. We elaborate on PDS schemes in Section 1.4 below.

As mentioned in Section 1.1, we "bootstrap trust" during refreshment phases by letting each node keep a small, unchanging public key in an unmodifiable ROM together with the code of the protocol. Specifically, we let each node keep a copy of the unchanging, public verification key of a PDS scheme in 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. At each refreshment phase, each node executes the following protocol:

- Choose a new pair of "personal" signature and verification keys of some "standard" (nondistributed) signature scheme. The new pair of keys should be chosen using "fresh" randomness. This randomness must *not* be the output of some pseudorandom generator whose seed was already known in previous time units. (This is in fact a property of most proactive solutions: the proactive refreshment phase must use fresh random choices.)
- 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 u is* v."

  The PDS scheme ensures that at most one certificate is generated for every node at every time unit. Hence, if N_i obtains a certificate in time unit u, it is guaranteed that the adversary cannot obtain a fake certificate with N_i's name on it in this round. A single instance of the PDS scheme is used to generate the certificates of all the nodes at all the time units.
- Finally, some work is done in order to maintain the PDS scheme itself. This includes obtaining fresh shares of the signing key, and *erasing* the old shares. See further discussion on the issue of data erasures in Section 6.

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 signing 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 ROM, and then verifies the senders 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.

In what follows we analyze this construction and show that it guarantees emulation and awareness properties (as discussed above) as long as the adversary is "not too powerful."

1.4. *Proactive Signatures Over Faulty Links*

All the PDS schemes known in the literature (e.g., [23], [20], [18], and [32]) are designed for a model where authenticated communication and broadcast primitives are available. The main technical contribution in this work is therefore to make these schemes work in the faulty-link model. This is done by constructing a "specialized version" of our authenticator, which takes any secure PDS scheme in the AL model and transforms it into a secure PDS scheme in the UL model. Note that most schemes in the literature assume a broadcast channel on top of reliable links, whereas our AL model does not provide such a channel. Yet, a broadcast channel can be emulated in the AL model using standard agreement protocols [31], [26], [27], [16], and [17].

Defining proactive signatures. To construct and prove a transformation such as above, we must have a definition for a "secure PDS." Unfortunately the literature so far does not contain any definition that suits our needs, so we start by presenting one in this work. Our definition is quite general, and applies also to threshold (non-proactive) signature schemes. It can also be regarded as a generalization of the notion of (centralized) signature schemes existentially secure against chosen message attacks [22]. (Gennaro et al. present two definitions of threshold distributed signature schemes in [20]. See more discussion on the relations between these definitions and ours in Section 3.4.)

Our definition again follows the usual paradigm for secure multiparty protocols. That is, we first formalize an ideal model for signature schemes. In this ideal model there are no cryptographic keys. Instead, an incorruptible trusted party keeps a database of signed messages. When enough signers wish to sign a message within a given time frame, the trusted party adds this message to the database. To verify whether a given message is signed, it suffices to query the database. Next we define a PDS scheme (either in the UL model or in the AL model) as one that emulates the ideal model process. The notion of emulation is the same as was discussed above for authenticators.

Realizing proactive signatures over faulty links. In order to transform a PDS scheme in the AL model to one in the UL model, we need reliability of the links (i.e., authenticity and delivery). We obtain delivery by a simple echo mechanism which, together with the assumed limitations of the adversary, ensures that enough messages arrive at their destinations. To get authenticity we apply essentially the same method as for the general proactive authenticator, using the PDS scheme itself to certify the local keys at the beginning of each refreshment phase. Our transformation does more work than is necessary for secure PDS schemes, to achieve extra properties that are needed to guarantee *awareness* of our proactive authenticator. (See the first remark in Section 4.3.3.)

1.5. *Road Map*

The rest of this paper is organized as follows. In Section 2 we formally define the two main computational models, namely the AL model and the UL model. We also define the central primitive of this paper: proactive authenticators. In Section 3 we define PDS schemes, both in the AL model and in the UL models. In Section 4 we show how to construct a PDS scheme in the UL model, given any PDS scheme in the AL model. In Section 5 we construct and prove the security of the proactive authenticator described above. Finally Section 6 contains discussions of several related issues.

A reader who is only interested in the PDS construction can skip most of the paper and only read Sections 4.1 and 4.2. For the definitions and security proof of this construction, one should read also Sections 2 (except 2.3), 3, and the rest of 4. The definition and construction of authenticators are found in Sections 2.3 and 5.

2. Models of Computation and Proactive Authenticators

Before defining proactive authenticators we describe the two underlying computational models. Both models postulate an adaptive, mobile adversary (i.e., an adversary that may break into different nodes at different points in time, where the decision to break into a particular node is based on the information gathered so far). One model (Section 2.1) assumes that the communication links are authenticated and reliable; the other model (Section 2.2) allows messages to be maliciously modified, deleted, and injected on the links. These two models are used in the definition of proactive authenticators (Section 2.3), as well as in the definition of PDS schemes (Section 3).

First we review some basic notions that underlie our formalization. A probability ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ is an infinite sequence of probability distributions, where a distribution $X(k,a)$ is associated with each values of $k \in \mathbf{N}$ and $a \in \{0,1\}^*$.

The distribution ensembles we consider in what follows describe computations where the parameter a describes the input and the parameter k is taken to be the security parameter. All complexity characteristics of our constructs are measured in terms of the security parameter. In particular, we will be interested in the behavior of our constructs when the security parameter tends to infinity.

Definition 1 (Statistical Indistinguishability). Two distribution ensembles X and Y are statistically indistinguishable (written $X \overset{s}{\approx} Y$) if for all $c > 0$, for all sufficiently large k and all a we have

$$\mathrm{SD}(X(k,a), Y(k,a)) < k^{-c},$$

where SD denotes statistical distance, or total variation distance (that is, $\mathrm{SD}(Z_1, Z_2) = \frac{1}{2}\sum_a |\mathrm{Prob}(Z_1 = a) - \mathrm{Prob}(Z_2 = a)|$).

Definition 2 (Computational Indistinguishability) [21], [35]. Two distribution ensembles X and Y are computationally indistinguishable (written $X \overset{c}{\approx} Y$) if for every algorithm D that is probabilistic polynomial-time in its first input, for all $c > 0$, all sufficiently

large k, all a, and all auxiliary information $w \in \{0,1\}^*$ we have

$$|\text{Prob}(D(1^k, a, w, X(k,a)) = 1) - \text{Prob}(D(1^k, a, w, Y(k,a)) = 1)| < k^{-c}.$$

Note that Definition 2 gives the distinguisher D access to an arbitrary auxiliary information string w (thus forcing the definition to be a nonuniform complexity one). It is stressed that w is independent from the random choices of X and Y.

2.1. *The Authenticated-Links (AL) Model*

We consider a *synchronous* network of n nodes, $N_1, \ldots, N_n$, where every two nodes are connected via a reliable communication channel. Each node N_i has input x_i and randomness r_i. The input x_i consists of pieces $x_i = x_{i,1}, x_{i,2}, \ldots$ where the piece $x_{i,w}$ is given to node N_i at the onset of communication round w. Also, to model the ability of nodes to make fresh random choices at refreshment phases, we think of r_i as consisting of pieces $r_i = r_{i,1}, r_{i,2}, \ldots$ where the piece $r_{i,w}$ is chosen by node N_i only at the onset of communication round w. In addition to x_i and r_i, each node is given n, the number of nodes, and a security parameter k (we let k be encoded in unary, to allow the nodes to be polynomial in their input length). A protocol π is the collection of programs run by the nodes.

The computation and adversary. The computation proceeds in rounds; in each round, except the first, each node receives all the messages sent to it in the previous round (these messages are received unmodified). It also gets its local input and randomness for this round. Next the node engages in an internal computation, specified by the protocol π. This computation may depend on its internal state, the messages it received, and the current time, and generates the outgoing messages and perhaps also some local output for this round.

A probabilistic polynomial time adversary, called an AL mobile adversary and denoted $\mathcal{A}$, gets as input the security parameter k and the number of nodes n (both encoded in unary), and interacts with the nodes as follows: $\mathcal{A}$ learns all the communication among the parties, and may also break into nodes and leave nodes at will. We say that a node is "broken" at a certain time, if the adversary broke into it before that time and did not leave it yet. When breaking into a node, N, the adversary learns the current internal state of that node. Furthermore, in all the rounds from this point and until it leaves N, the adversary may send messages in the name of N, and may also modify the internal state (i.e., the memory contents) of N. The only restriction is that N may have some Read-Only Memory (ROM), which is fixed at the onset of the protocol and cannot be changed afterwards, not even by the adversary. Typically, this is where the protocol code itself is stored. (In the UL model we also allow an additional small part of the memory to be ROM.) When the adversary leaves a node N, it can no longer send messages in N's name, and has no more access to the internal state of N. In particular, the adversary does not see the random choices of N in forthcoming rounds.

We allow rushing. That is, in each communication round the adversary first learns the messages sent by the nonbroken nodes, and only then generates the message of the broken nodes. In addition, the adversary can choose to break into new nodes after learning the messages sent by each node.

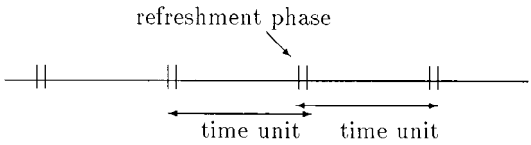

Fig. 1. Time units and refreshment phases.

We assume an initial set-up phase where the parties communicate without the intervention of the adversary. That is, during this phase the adversary does not break into parties and does not learn the messages sent on the links. Typically, the set-up phase is used to choose and exchange cryptographic keys. We remark that the set-up phase can be replaced by an execution of a centralized set-up algorithm. We chose the distributed formalization since it somewhat simplifies the syntax later on.

Time units, refreshment phases, and the power of the adversary. We divide the lifetime of the system into time units and 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. (The reason for the name is that nodes typically "refresh their cryptographic keys" during this overlap.) See Fig. 1. Typically, the duration of a refreshment phase is several communication rounds (i.e., up to a few seconds), where a time unit may last hours, days, or even months.

Definition 3 (t-Limited Adversary). An AL-model adversary $\mathcal{A}$ is called t-limited with respect to protocol π if, on any inputs and random inputs for the parties and adversary, $\mathcal{A}$ breaks into at most t nodes at each time unit.

When the protocol π is clear from the context, we sometimes omit it and just say that $\mathcal{A}$ is t-limited. It should be noted that even a 1-limited adversary can break into all nodes at one point or another, as long as at most one node is broken into at each time unit.

Executions, transcripts, and outputs. An execution of a protocol π with security parameter k, input vector $\vec{x}$, adversary $\mathcal{A}$, and randomness $\vec{r} = r_A, r_1, \ldots, r_n$ (r_A for $\mathcal{A}$, and r_i for node N_i) is the process of running π, $\mathcal{A}$ with inputs $\vec{x}$ and randomness $\vec{r}$ as described above. The transcript of this execution, denoted $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$, records all the information relevant to this execution. This includes the inputs and randomness of the nodes and adversary, all the messages sent on the links, and the local outputs of all nodes and adversary. In particular, the transcript of an execution uniquely determines the outputs of all the parties and the adversary from this execution. Let $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x})$ denote the random variable having the distribution of $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$ where $\vec{r}$ is uniformly chosen in its domain. In what follows we often identify an execution with its transcript. (We use the notion of transcripts only in our proofs of security in Sections 4 and 5.)

The global output of an execution contains only the information that is relevant for the *functionality* of the protocol with regard to the external world. Roughly, it includes the local outputs of the parties (as specified by the protocol π), together with the adversary's

output and some other relevant data. More precisely, in each communication round, a node that is not broken into outputs whatever is instructed by its protocol, broken nodes have empty output, and the adversary may also have output of its own. Also, whenever a node N_i is broken into, the line "`Node` N_i `is compromised`" is added to N_i's output. The line "`Node` N_i `is recovered`" is added to N_i's output when the adversary leaves N_i. (We stress that a node does not necessarily "know" when it is broken or recovered. The above notices can be thought of as part of a "system log" that is added *externally* to the node's output in order to better capture the functionality of the protocol.)

The global output of the computation up to round w is the concatenation of the local outputs from all rounds of all the nodes and adversary. When dealing with protocols for specific tasks, it may be convenient to let the global output contain additional information that is relevant to this task. For example, in the definition of PDS schemes, the global output contains information about whether messages have valid signatures or not. See Section 3.

Let $\text{AL-OUT}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$ denote the global output of the execution with transcript $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$. It is convenient to think of the global output as an $n + 1$ vector where the zeroth component contains the adversary's output and the ith component contains the output of N_i. Let $\text{AL-OUT}_{\pi,\mathcal{A}}(k, \vec{x})$ denote the random variable describing $\text{AL-OUT}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$ where $\vec{r}$ is uniformly chosen from its domain. Let $\text{AL-OUT}_{\pi,\mathcal{A}}$ denote the probability ensemble $\{\text{AL-OUT}_{\pi,\mathcal{A}}(k, \vec{x})\}_{k\in\mathbf{N},\vec{x}\in\{0,1\}^*}$.

2.2. *The Unauthenticated-Links (UL) Model*

The UL model is similar to the AL model, except for the following two modifications. First, in addition to its capabilities in the AL model, here the adversary (called a UL mobile adversary and denoted $\mathcal{U}$) may modify, delete, and inject messages sent on the links. That is, as in the AL model, at the end of each round each node sends messages to other nodes. Yet here it is the adversary that decides on the values of the messages received by the nodes at the beginning of the next round. These values need not be related in any way to the values that were sent. (We note, however, that an adversary which is "not too powerful" delivers most messages without changing them. See discussion below.) Second, we allow the nodes to write to a special small ROM at the end of the set-up phase. As explained in the Introduction, this provision is what makes a reasonable solution possible.

An execution of a protocol π with random inputs $\vec{r}$, input vector $\vec{x}$, and a UL adversary $\mathcal{U}$ is defined in a similar manner to the AL model, except that here the adversary has the additional capabilities described above. (We stress that the adversary remains inactive during the set-up phase.) Transcripts are defined as in the AL model. We let $\text{UL-TRANS}_{\pi,\mathcal{U}}(k, \vec{x}, \vec{r})$ and $\text{UL-TRANS}_{\pi,\mathcal{U}}(k, \vec{x})$ have analogous meaning, in the UL model, to $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x}, \vec{r})$ and $\text{AL-TRANS}_{\pi,\mathcal{A}}(k, \vec{x})$. The definition of the global output will depend on a parameter, defined below, used to measure the power of the adversary in controlling the communication links. We thus turn to defining the power of the adversary.

The power of a UL adversary. Capturing the type of adversaries that our constructions withstand in the UL model takes some care. Here 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 and connectivity of communication links it disrupts. To make this precise, we start with a definition of reliable links.

Definition 4 (Reliable Links). Let E be an execution of a protocol π with adversary $\mathcal{U}$. A link between nodes N_i and N_j is reliable during a certain time interval in E, if during this time:

(a) Neither N_i nor N_j is broken into.
(b) Every message sent on this link arrives unmodified at the other node at the end of the communication round in which it was sent. Also, N_i, N_j do not receive any other message on this link.

(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.) If either (a) or (b) does not hold, we say that the link between N_i and N_j is unreliable.

We comment that since this is a synchronous model, we do not worry about ordering of messages (without loss of generality a party sends only one message to each other party at each round, and all the messages that were sent at a round arrive together at the beginning of the next round). Also, the definition above excludes "replay attacks," since a replayed message is considered "another message," and hence it is excluded by condition (b) above.

At any time during an execution of a protocol π with adversary $\mathcal{U}$, we distinguish between three categories of nodes in the network: broken, disconnected, and operational. Broken nodes are those broken into by the adversary. Defining disconnected and operational nodes is more subtle. In principle, we would like to have a parameter s, so that a node is disconnected if it has at least s unreliable links, and is operational otherwise. Formalizing this notion in a meaningful way requires some care, however.

Recall that nodes use their reliable links to regain their security after a break-in. Consider now a set of nodes that were broken in time unit $(u-1)$ but are not broken in time unit u, and that have reliable links to one another during time unit u. Although these nodes are not broken and have "many reliable links" in this time unit, they may still be unable to recover from a break-in. The reason is that they may only be able to communicate with nodes that were also broken in the previous time unit, and thus may have no one that can actually help them to recover. We therefore formulate the notion of an operational node in an inductive manner, making sure that a node that is operational in time unit u is able to communicate with nodes that were operational in time unit $u-1$.

Definition 5 (Operational Nodes). Let E be an execution of an n-node protocol π with UL adversary $\mathcal{U}$, and let $s \leq n$. For each communication round in this execution, the set of s-operational nodes during this round is defined inductively as follows:

1. In the first communication round of the first time-unit, the s-operational nodes are all those that are not broken.

2. A node which was s-operational at the previous communication round, remains s-operational at the current round, provided that:
   (a) It was not broken at this round.
   (b) During this round it has reliable links with at least $n - s + 1$ nodes that were also s-operational at the previous communication round. (Alternatively, it has unreliable links to less than s other s-operational nodes.)
3. A node, N, which was *not* s-operational at the beginning of time unit u, becomes s-operational at the end of the refreshment phase of this time unit only when:
   (a) It was not broken throughout this refreshment phase.
   (b) There is a set S, consisting of at least $n - s + 1$ nodes that are s-operational throughout this refreshment phase, such that N has reliable links with all the nodes in S throughout this refreshment phase.

We note that a 1-operational node has good links to all other nodes, whereas an n-operational node only has a good link to just one other node. This somewhat counter-intuitive terminology will become convenient in the proof of our main theorem (Section 4.3).

Definition 6 (Disconnected Nodes). A node is said to be s-disconnected at a certain communication round in time unit u if it is not broken but also not s-operational in this communication round. (Intuitively, a node is s-disconnected if it has s or more unreliable links.)

We remark that just like the broken nodes, the identities of the disconnected nodes at each point during the execution are determined by the actions of the adversary up to this point. One difference between broken and disconnected nodes, however, is that being broken is a zero/one situation (a node is either broken or not) while being disconnected is parameterized. Namely, an $(s+1)$-disconnected node is "more disconnected" than an s-disconnected node.

Still, for the purpose of defining the power of the adversary, we need to count how many nodes are impaired by the adversary, so we set a threshold s such that a node is considered disconnected if it is at least s-disconnected. Thus the power of the adversary is defined by means of two parameters: the "disconnection threshold" s and a bound t on the number of nodes "impaired" by the adversary in every time unit. That is:

Definition 7 ((s,t)-Limited Adversary). Consider an execution of π with the UL adversary $\mathcal{U}$. We say that $\mathcal{U}$ is (s,t)-limited in this execution if during every time unit, at most t nodes are either broken or s-disconnected. $\mathcal{U}$ is (s,t)-limited with respect to π if it is (s,t)-limited in any execution with nodes running π.

It is stressed that in order for an adversary to *not* be (s,t)-limited, it must "attack" at least t nodes *at the same time unit*, where attacking a node means either breaking into it, or actively modifying the communication on s of its links. This may be a difficult task even when s and t are even modestly large.

The output of an execution. We can now return to defining the global output of an execution. The global output of an execution is similar to the AL model, except that here, the line "`Node` N_i `is compromised`" is added to N_i's output whenever it stops being s-operational. (This happens not only when it is broken into, but also when it becomes s-disconnected.) Similarly, the line "`Node` N_i `is recovered`" is added to N_i's output only when it becomes s-operational again. (Note that, since the definition of the protocol's output depends on the parameter s, one can have different definitions for the output of the protocol, depending on what properties one wants to prove about this protocol.) We let $\text{UL-OUT}_{\pi,\mathcal{U},s}(k,\vec{x},\vec{r})$ and $\text{UL-OUT}_{\pi,\mathcal{U},s}(k,\vec{x})$ have analogous meaning, in the UL model, to $\text{AL-OUT}_{\pi,\mathcal{A}}(k,\vec{x},\vec{r})$ and $\text{AL-OUT}_{\pi,\mathcal{A}}(k,\vec{x})$ from above, with respect to s-operational nodes. Let $\text{UL-OUT}_{\pi,\mathcal{U},s}$ denote the probability ensemble $\{\text{UL-OUT}_{\pi,\mathcal{U},s}(k,\vec{x})\}_{k\in\mathbf{N},\vec{x}\in\{0,1\}^*}$.

In the next section, where we define authenticators, it is convenient to have a single parameter (rather than two) to describe the power of the adversary in the UL model. Hence, the definitions below use Definition 7 with $s = t$. Similar definitions can be made for other settings of s and t, but we do not discuss them in this paper.

2.3. *Proactive Authenticators*

This section defines the requirements from a "compiler" that transforms protocols that assume a network with authenticated links (and repeated break-ins) into protocols that can run over a network with *un*authenticated links (and repeated break-ins). We call such compilers proactive authenticators.

In the text below we ignore many syntactic issues related to "how do you describe a protocol?" We assume some standard way of describing protocols, and define authenticators as procedures that operate on such descriptions. An authenticator takes for input a description of a protocol in the AL model, and outputs a description of another protocol (for the same number of nodes) in the UL model. Our main security requirement from such a compiler, called emulation, roughly means that any protocol in the AL model is transformed into a protocol in the UL model with essentially the same functionality. This requirement follows the general paradigm of defining secure multiparty protocols [28], [1], [10]. We complement this requirement with an alternative one, called awareness, that means that a node will notice whenever it is being "impersonated" by the adversary.

Definition 8 (Emulation). Let π and π' be protocols in the AL and UL models, respectively. We say that π' t-emulates π in the UL model if for any (t,t)-limited UL adversary $\mathcal{U}$ there exists a t-limited AL adversary $\mathcal{A}$ such that

$$\text{AL-OUT}_{\pi,\mathcal{A}} \overset{c}{\approx} \text{UL-OUT}_{\pi',\mathcal{U},t}. \tag{1}$$

Note that requirement (1) incorporates many conditions. In particular, the combined distributions of the outputs of the parties, the adversary's output, and the identities of broken nodes, should be indistinguishable on the two sides of (1). In general, this condition captures the required notion of "security equivalence" between the protocols in the sense that any consequences of the actions of the strong UL adversary against nodes running protocol π' can be imitated or achieved by the weaker AL adversary against nodes running protocol π, without requiring breaking into more (or different) nodes. See [10] for more discussion.

Definition 9. An authenticator Λ is t-emulating if, given any protocol π in the AL model, Λ generates a protocol $\Lambda(\pi)$ that t-emulates π in the UL model.

Awareness. As explained in the Introduction, the emulation property allows a limited number of nodes to be disconnected from the rest of the network, and consequently be impersonated by the adversary. This is an inevitable characteristic of our model. Yet, we can guarantee that a node will be *locally* aware of the fact that it is being thus attacked. Such a guarantee may still be valuable; in particular, if the nodes operate in an environment where a higher level protocol or a human operator can use out-of-band communication to restore security, then detecting an attack may be nearly as good as preventing it.

Before we define awareness, we must first define what it means for the adversary to "impersonate" a node in the network in the eyes of other nodes. To this end, we restrict ourselves to a special type of authenticators, which we call layered authenticators. Given a protocol π, a layered authenticator Λ generates a protocol $\Lambda(\pi)$ that consists of two modules, called layers. At the top layer π runs unchanged. At the bottom layer, for each message that π instructs to send and receive, the node follows some procedure specified by Λ. In addition, in the system set-up and in each refreshment phase the nodes also execute some refreshment protocol specified by Λ.

Definition 10 (Internal and External Views, Impersonation). Let Λ be a layered authenticator and let π be an n-node protocol, and consider an execution of $\Lambda(\pi)$ in the presence of a UL model adversary $\mathcal{U}$. Consider a time unit u within this execution and let N_i be a node that is not broken during time unit u.

The internal view of N_i during u is the sequence of all the messages that are sent and received by the top layer (i.e., by protocol π) during u.

The external view of N_i in time unit u consists of all the messages that appear, in the internal views of other nonbroken nodes in the network, to be received from N_i during u.

We say that a node N_i is being impersonated at time unit u if its external view contains a message which is not in its internal view.

We stress that a node N is not impersonated when messages that it sends do not arrive at their destinations. It is only impersonated when messages that it does not send are accepted by other nodes as authentic messages coming from N.

Definition 11. A layered authenticator Λ is (s, t)-aware if any protocol generated by Λ satisfies the following property: For any (s, t)-limited UL adversary $\mathcal{U}$, every impersonated node outputs a special alert signal in each time unit in which it is impersonated, except with a negligible probability in the security parameter k.

(A function δ: $\mathbf{N} \to [0, 1]$ is negligible if for all $c > 0$ and all large enough k we have $\delta(k) < k^{-c}$.)

Naturally, we want protocols that output `alert` only with "good reason." This is guaranteed by the emulation requirement: Since the `alert` output never appears in a global output in the AL model, we are guaranteed that in a protocol generated by a t-emulating authenticator Λ no t-operational node outputs `alert` as long as the UL

adversary is (t, t)-limited. At the same time, the definition of t-emulation allows nodes that are not t-operational in the UL model to output `alert`, since the global output treats a node that is not t-operational as broken, even if this node is only t-disconnected.

3. Defining Proactive Distributed Signature (PDS) Schemes

Our main technical tool is a transformation from any "secure PDS scheme in the AL model" into a "secure PDS scheme in the UL model." However, before we can present the construction we first need to define secure PDS schemes. Below we present a general paradigm for defining "secure distributed signature schemes." This paradigm generalizes the notion of security against existential forgery under chosen message attack [22]: even after seeing signatures on messages of its choice, an adversary should be unable to come up with any new message and a valid signature on this message. We use this general paradigm to define secure PDS schemes in the AL and UL models. The general paradigm, as well as the particular definitions, may well be of independent interest.

We follow the general framework of defining secure distributed protocols. That is, we describe an ideal process that captures the required functionality from a distributed signature scheme; next we define a secure scheme as one which emulates the ideal process in the sense of Section 2.3. This approach does not require a distributed signature scheme to be based on any "traditional," or centralized, signature scheme. Yet, when reduced to the special case of centralized signature schemes, the definition below coincides with the one in [22]. At the end of this section (Remark 7) we briefly discuss the alternative approaches taken in [20] toward defining secure distributed signature schemes.

In Section 3.1 we describe the ideal model for PDS schemes. In Section 3.2 we describe some syntax related to PDS schemes in the AL and UL models, and in Section 3.3 we present the definition. Finally, in Section 3.4 we discuss some of our choices.

3.1. *The Ideal Process*

We first describe the ideal process for distributed (threshold) signatures. We note that the only *functionality* we care about in a signature scheme is that potential verifiers are able to distinguish between messages that were properly signed and messages that were not. More precisely, we need the following functionality:

Threshold. A message is signed only if enough signing parties (called signers) agree to sign it.

Correctness. A message is verified only if it is signed.

Public verifiability. The status of a message can be verified without the participation of any of the signers.

We view the signatures attached to messages and the keys used to verify these signatures merely as tools for obtaining this functionality. Consequently, cryptographic keys and signatures do not appear in the ideal process. Instead, in the ideal process there is a trusted party that keeps a database of "signed messages." Initially the database is empty. The trusted party inserts a message to the database once it is asked to do so by an appropriate subset of the signers. Verification is done simply by checking whether or not a message appears in the database.

More precisely, in the ideal process there are n signers $N_1 \cdots N_n$ and a signature verifier V. (V is used to ensure that signed messages can be verified without interacting with the signers.) The parties communicate with a trusted party T and a probabilistic-polynomial-time ideal-model forger $\mathcal{IF}$. There is no communication among the signing parties. The process is parameterized by t, the threshold for signing messages, and by a security parameter k. (The role of the security parameter in the ideal model is to bound the running time of the ideal-model forger: since the forger is a polynomial-time algorithm, its running time is bounded by some polynomial in k.) In addition, the forger and the signers may have external inputs; these inputs are not used directly by the signing parties; their goal is to capture external information that the forger may have (much like auxiliary input for general protocols). The interaction proceeds as follows:

1. Initially, the forger $\mathcal{IF}$ is given k and n (encoded in unary). The signing parties, the verifier, and the trusted party have no input. The trusted party initializes the set of signed messages $M \leftarrow \emptyset$.

   Also, all the involved parties have access to a common variable called the time unit counter and denoted u. (As seen below, the counter is meant to capture the synchrony of the network.) Initially u is set to 1, and the adversary can increment u at wish.
2. The adversary invokes signers of its choice with arbitrary values (these values represent messages to be signed.) When a signer N_i gets a value m from the adversary at time unit u, it sends a request "`sign` (m, u)" to the trusted party T, and appends the line "N_i `is asked to sign` m `at time unit` u" to its output.
3. Once the trusted party T receives "`sign` (m, u)" requests from at least $t+1$ signers, and these requests agree on m and u, it adds (m, u) to the set M of signed messages. Every signer that sent a "`sign` (m, u)" request, receives a response "(m, u) `is signed`" from T, and appends this response to its output. (Note that this formalization forces all requests to sign a message to arrive at the same time unit).
4. The adversary may break into a signer N_i at any time by invoking it with a "break-in" input. The effect here is that the line "N_i `is compromised`" is appended to N_i's output, and the forger learns the signers input. Also, from this point and until the adversary leaves N_i, the output of N_i is under the control of the adversary.

   The adversary can leave a broken node at any time by invoking it with a "recover" input. Here the line "N_i `is recovered`" is appended to the N_i's output, and N_i then resumes outputting values as in Steps 2 and 3.
5. The adversary may query the verifier V with a message m. If $m \in M$ then V responds with "m `is verified`" and appends a similar line to its output. Otherwise V responds with "m `is not verified`," but *does not add anything to its output*. See Remark 2 at the end of this section for a discussion on this detail.
6. The interaction ends when the adversary halts. The output of the interaction is the concatenation of the outputs of the adversary, the n signers, and the verifier.

Output. The $(n+2)$-vector

$$\begin{aligned}\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r}) \;=\; &\langle \text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_A, \text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_1 \\ &\cdots \text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_n, \text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_V \rangle\end{aligned}$$

denotes the global output of an execution of the system above and the adversary $\mathcal{IF}$ on randomness $\vec{r}$, input $\vec{x} = x_F, x_1, \ldots, x_n$ for the forger and signers, threshold t, and security parameter k. (The adversary output is $\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_A$, the output of signer N_i is $\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_i$, and the verifier's output is $\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})_V$.) Let $\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x})$ describe the distribution of $\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x}, \vec{r})$ when $\vec{r}$ is uniformly distributed over its domain. Let $\text{ID-SIG}_{t,\mathcal{IF}}$ denote the ensemble $\{\text{ID-SIG}_{t,\mathcal{IF}}(k, \vec{x})\}_{k\in\mathbf{N},\vec{x}\in\{0,1\}^*}$. See Remark 1 in Section 3.4 for discussion on the need for including the adversary output in the global output.

3.2. *The Structure and Operation of PDS Schemes*

We now describe the syntax and operation of PDS schemes, either in the AL or the UL model.

Syntax. A PDS scheme has four components: a key generation protocol Gen, a distributed signing protocol Sign, a verification algorithm Ver, and a distributed refresh protocol Rfr.

- In the key-generation protocol, Gen, each node is given a security parameter k (encoded in unary), and it outputs a pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. (The intent is that the pk_i's are the same for all the nodes, and that the sk_i's are shares of the corresponding secret key.)
- In the signing protocol Sign, each signing node N_i is given as input a triple $(\mathsf{pk}, \mathsf{sk}_i, m)$, plus perhaps some additional information, and outputs a value σ. (The intent is that σ is a signature on m, verifiable by the public key pk.)
- The verification algorithm Ver is given as input a triple (pk, m, σ). It outputs a binary *pass*/*fail* value. We say that σ is a valid signature on m with respect to public key pk if $\mathsf{Ver}(m, \sigma, \mathsf{pk}) = \mathit{pass}$.
- In the refresh protocol Rfr, each signing node N_i is given as input a pair $(\mathsf{pk}, \mathsf{sk}_i)$, and it outputs a value sk'_i. (The intent is that sk'_i is the new share of the secret key.)

Operation. A PDS scheme proceeds as follows, in both the AL and UL models. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Rfr})$ be an n-node PDS scheme and let $\mathcal{F}$ be a forging adversary (in either of these models). An execution of $\mathcal{S}$ with $\mathcal{F}$ consists of the following process, involving n signers $N_1, \ldots, N_n$, the adversary $\mathcal{F}$, and an (unbreakable) signature verifier V. (As in the ideal model, the signature verifier captures public verifiability of the generated signatures. Its only role is to run the public verification algorithm on given (*message*,*signature*) pairs.) Also here, the nodes may have (auxiliary) input, which they ignore.

Set-up phase. The nodes $N_1 \cdots N_n$ execute the key-generation protocol Gen. During this phase the adversary cannot break any node or interrupt the communication, and it also does not learn the communication.

We denote the output of node N_i by $(\mathsf{pk}, \mathsf{sk}_i)$. Note that since this phase is executed in an "adversary free" environment, we can assume that at the end of this protocol all the nodes indeed have the same public key, pk. Below we also assume, without loss of generality, that k is implicit in the keys generated by $\mathsf{Gen}(k)$.

Once the execution of Gen is terminated, the adversary in given the inputs n, k, t as well as public key pk.

Signatures. The adversary (in either model) can send a special message "`sign` m" to any of the signing nodes. Upon receipt of this message, N_i outputs "N_i `is asked to sign` m `at time unit` u" where u is the current time unit, and runs protocol Sign on input $(\mathsf{pk}, \mathsf{sk}_i, \langle m, u\rangle)$. This protocol is executed in either the AL or the UL model. If at the end of the execution of Sign, N_i obtains a valid signature on $\langle m, u\rangle$ (with respect to pk), then the line "$\langle m, u\rangle$ `is signed`" is added to its output.

Verification. The adversary $\mathcal{F}$ can send to the signature verifier V a pair (μ, σ). The verifier V runs algorithm Ver on $(\mathsf{pk}, \mu, \sigma)$; if the verification succeeds, then V outputs "μ `is verified`." If the verification fails then it outputs nothing (see Remark 2 in Section 3.4). We stress that V cannot be broken into.

Refreshment. In the refreshment phase at the beginning of each time unit u, the nodes run protocol Rfr (either in the AL or UL model). Node N_i invokes Rfr on input $(\mathsf{pk}, \mathsf{sk}_{i,u-1})$ and a fresh random input $r_{i,u}$. (We let $\mathsf{sk}_{i,0} = \mathsf{sk}_i$.) The value of $\mathsf{sk}_{i,u}$ is set to the output of Rfr. Once $\mathsf{sk}_{i,u}$ is calculated, N_i *erases* the values $\mathsf{sk}_{i,u-1}$ and lets $\mathsf{sk}_{i,u}$ replace sk_i in protocol Sign.

The execution terminates when the adversary $\mathcal{F}$ halts. The output of each party is the concatenation of all its intermediate outputs up to that point. The global output of the execution is the concatenation of the outputs of the adversary $\mathcal{F}$, the signers $N_1, \ldots, N_n$ and the verifier V.

Notation. The $(n+2)$-vector

$$\begin{aligned}\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r}) \;=\; &\langle \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_A, \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_1 \\ &\cdots \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_n, \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_V\rangle\end{aligned}$$

denotes the global output of an execution of the scheme $\mathcal{S}$ with the AL-model adversary $\mathcal{AF}$ on randomness $\vec{r}$, inputs $\vec{x} = x_{\mathrm{F}}, x_1, \ldots, x_n$ for the forger and signers, and security parameter k. (The adversary output is $\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_A$, the output of signer N_i is $\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_i$, and the verifier's output is $\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_V$.) We denote by $\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x})$ the distribution of $\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})$ when $\vec{r}$ is uniformly distributed.

Ensembles $\text{UL-SIG}_{S,\mathcal{UF}}$ and $\text{UL-SIG}_{S,\mathcal{UF}}$ are defined similarly with respect to a UL-forger $\mathcal{UF}$ and scheme S.

3.3. *Security of a PDS Scheme*

Security is defined via emulation of the ideal signature process. Namely, a PDS scheme is deemed secure if any adversarial behavior against it can also be carried out in the ideal model.

Definition 12. An n-node PDS scheme S is t-secure in the AL model if for any t-limited AL-forger $\mathcal{AF}$ there exists an ideal-model forger $\mathcal{IF}$ such that

$$\text{AL-SIG}_{S,\mathcal{AF}} \overset{\text{c}}{\approx} \text{ID-SIG}_{t,\mathcal{IF}}. \tag{2}$$

S is (s,t)-secure in the UL model if for any (s,t)-limited UL-forger $\mathcal{UF}$ there exists an ideal forger $\mathcal{IF}$ such that

$$\text{UL-SIG}_{S,\mathcal{UF}} \overset{c}{\approx} \text{ID-SIG}_{t,\mathcal{IF}}. \tag{3}$$

Remark 4 below discusses the relations between the PDS threshold and the power of the adversary.

3.4. *Discussion*

Remark 1: *The structure of the global output.* As in the case of Definition 8, the requirement that the global output of the real-life computation be distributed indistinguishably from the output of the global output of the ideal process implies many conditions. In particular, it implies that in the real-life computation the verifier does not accept a message as signed, unless at least $t+1$ signers were sent a "`sign` m" message. It also implies that whenever at least $t+1$ "good nodes" invoke the signature protocol on the input m, then a signature is obtained and becomes publicly available.

Intuitively, there is no real need to incorporate the output of the adversary in the global output, since we do not care about information gathered by the adversary during the computation, as long as this information does not help the adversary to forge signatures. Indeed, it can be readily seen that removing the adversary output from the global output does not affect the definition. (That is, a definition that is identical to Definition 12 except that the the global output does not include the adversary output, is equivalent to Definition 12, in both the AL and UL models.) We choose to include the adversary output in the global output in order to conform with the format of Definition 8. In particular, this choice becomes useful in Section 5, where we show that the transformation from a PDS scheme in the AL model to one in the UL model is general enough to be a proactive authenticator.

Remark 2: *The verifier's output.* We motivate our choice to have the verifier V output nothing when a verification of some signature fails: Assume that in such a case V outputs a failure message (in all models). Then the following trivial (and harmless) adversarial behavior cannot be emulated in the ideal model. The adversary sends to V a pair (m,σ) where m is a message that was legally signed, but σ is *not* a valid signature for m. Now, in the ideal model V would output that m is OK since it is in the database of signed messages, but in both the AL and the UL models V would reject m. By having V output nothing in case of failure we allow the ideal-model forger to decide whether to query V with m, based on whether in the real-life model the forger queried V with a correct signature.

On a more abstract level, this provision captures an inevitable weakness of digital signatures relative to the ideal process: using digital signatures, a verifier cannot tell the difference between the case where a document was never signed and the case where the document was signed but the presenter of the document simply failed to present a valid signature.

Remark 3: *Public verifiability.* By making the database in the ideal model completely public, Definition 12 models a setting where *everyone* can verify whether a given message

is signed. This simple scenario is sufficient for the purpose of this paper. Yet we note that digital signatures may be used also in scenarios where it is important that only holders of a valid signature should be able to verify that a message is signed. Such scenarios may be modeled via ideal processes that are different from the one here.

Remark 4: *The threshold values*. Recall that a PDS scheme includes a threshold t' (where at least $t' + 1$ nodes are needed to generate a signature). The various models of computation, on the other hand, include some measure t for the power of the adversary, which is the number of nodes it can compromise in any time unit. Clearly, we must have $t' + 1 > t$, or else the adversary will be able to generate signatures. Similarly, we must have $t' + 1 \leq n - t$, so that nodes that are not compromised will be able to generate signatures. (The combination of these two requirements implies that the bound $t < n/2$.) In Definition 12 above we set $t' = t$. Although there may be settings in which it makes sense to set the threshold t' to some other value between t and $n - t - 1$, this extra generality is not really needed in this paper.

Remark 5: *Time granularity.* In order to generate a signature on a message m, our definition requires that the adversary's requests to sign m are made *within a single time unit*. This particular "granularity of time" is convenient in the proactive setting. In other settings different granularity may be required (for instance, it may be required that all requests are made within the same communication round).

Remark 6: *Postexecution corruptions*. General definitions of security of protocols in the presence of an adaptive adversary make the following additional requirement from a secure protocol [1], [10]: The ideal-model adversary should be able to handle corruption requests made *after* the protocol execution is completed. (These requests may be made when the adversary interacts with other protocols run by the party.) Here we do not need to make this additional requirement, since the PDS scheme remains active throughout the execution of the system.

Remark 7: *Alternative definitions*. Gennaro et al. present two definitions of security of threshold (not proactive) signature schemes in a setting where the communication links are authenticated and a *broadcast* channel exists [20]. One definition, called unforgeability, essentially requires that, even after interacting with the signers, the adversary be unable to generate signatures unless sufficiently many signers have agreed to sign the message. In addition, a collaboration of sufficiently many signers should always be able to generate signatures. This definition is essentially *equivalent* to our definition, when formulated in their setting. (Yet, some extra work is needed for formulating the [20] unforgeability definition in our unauthenticated setting.)

Their other definition, called simulatability, proceeds roughly as follows. Start from some particular centralized signature scheme (in the case of [20] this is the DSS scheme), and regard its key-generation and signing algorithms as probabilistic functions. Next, require that the distributed key-generation and signing protocols of the threshold scheme securely evaluate the corresponding centralized functions, according to some standard definition of secure multiparty function evaluation. Informally, one is now guaranteed that "the threshold scheme mimics whatever properties the centralized scheme has."

This approach allows discussing schemes that have weaker properties than existential security against chosen ciphertext attacks. That is important in their case, since the DSS scheme, and consequently their construction, is not known to be existentially secure against chosen message attack under standard assumptions. Yet this approach is somewhat indirect in nature since it bases itself on some specific centralized scheme and does not explicitly require any unforgeability properties. In addition, the secure function evaluation requirement is quite strong. In particular, there exist threshold schemes that are unforgeable but do not securely evaluate *any* function whatsoever.

4. Constructing PDS Schemes in the UL Model

We use the following basic result as our starting point:

Theorem 13. *If trapdoor permutations exist, then for any $n \geq 2t+1$ there exist n-node t-secure PDS schemes in the AL model.*

The theorem can be proven using generic techniques from the literature. One may start from any secure (centralized) signature scheme. Such schemes can be realized using specific computational assumptions such as factoring (e.g., [22] and others), any trapdoor permutation [3], [29], or even any one-way function [34]. This centralized scheme can be transformed into a threshold scheme (assuming secure channels and a broadcast channel), using techniques for secure multiparty computations [6], [33]. Next, the threshold scheme can be made proactive, again using techniques for secure multiparty computations [30]. Finally, the secure channels can be implemented using noncommitting encryption [11], and the broadcast channel can be implemented using an agreement protocol, such as the one in [16]. The result is a (rather inefficient, but still polynomial-time) secure PDS scheme in the AL model.

Furthermore, recent works show how to construct more efficient secure PDS schemes in the AL model from many centralized signature schemes such as El-Gamal, DSS, and RSA [23], [20], [18], [32]. (In order to use these schemes in our construction one has to *assume* that the underlying centralized signature schemes are existentially unforgeable against chosen message attacks. Sometimes this assumption can be supported by analyses in idealized models, such as the *random oracle model*.)

In this section we show that under the same conditions, there also exist n-node (t,t)-secure PDS schemes in the UL model. Specifically, we show that any n node t-secure secure PDS scheme in the AL model (with $n \geq 2t+1$) can be transformed into an n node (t,t)-secure PDS scheme in the UL model.

More precisely, let $\mathsf{ALS} = \langle A\mathsf{Gen}, A\mathsf{Sign}, A\mathsf{Ver}, A\mathsf{Rfr}\rangle$ be an n node t-secure PDS schemes in the AL model, as in Definition 12. In addition we use a centralized signature scheme, $\mathsf{CS} = \langle C\mathsf{Gen}, C\mathsf{Sign}, C\mathsf{Ver}\rangle$, that is existentially unforgeable under an adaptive chosen message attack as in [22]. Given schemes ALS and CS, we construct an n node PDS scheme in the UL model, denoted $\mathsf{ULS} = \langle U\mathsf{Gen}, U\mathsf{Sign}, U\mathsf{Ver}, U\mathsf{Rfr}\rangle$, and show that ULS is (t,t)-secure:

Theorem 14. *If* ALS *is an n node t-secure PDS scheme in the AL model with $n \geq 2t+1$, and* CS *is a centralized signature scheme existentially unforgeable under an adaptive*

chosen message attack, then the resulting scheme ULS *is an n node* (t, t)*-secure PDS scheme in the UL model.*

The transformation of ALS to ULS uses the idea of a *layered authenticator* (see Section 2.3). That is, the scheme ULS can be thought of as having two layers: At the top layer we have ALS running unchanged. At the bottom layer the nodes follow some protocol for each message that ALS instructs to send and receive, and in addition take some steps during each refreshment phase.

The crux of the construction is the protocol executed in the proactive refreshment phases. Here the nodes "bootstrap" their trust in their memory and their ability to authenticate communicated data. This "bootstrapping" process consists of two parts: First each node chooses a new pair of signing and verification keys of scheme CS, and tries to obtain a certificate, verifiable using the verification key of ALS, to its newly chosen verification key of CS. This certificate is generated by the nodes using their signing keys for ALS *from the previous time unit*. At the second part of the "bootstrapping" process the nodes generate their signing keys of ALS for the new time unit, and *erase* their old keys.

The rest of this section is organized as follows: In Section 4.1 we describe some tools used in the construction. These are simple protocols that obtain "somewhat reliable communication" over unauthenticated links. In Section 4.2 we present the scheme ULS, and in Section 4.3 we prove the security of this scheme in the UL model.

4.1. *Communicating over Faulty Links*

Below we describe three communication protocols used in our construction of PDS in the UL model, give some intuition for their use, and prove a few basic properties about them.

To help the intuition, we use the suggestive names "accept" and "receive" in the description below to describe receipt of messages: intuitively, a node "receives" a message if this message was delivered to it, and it "accepts" the message if it believes that this message is authentic.

Connectivity. We start with a protocol to guarantee that the adversary must disrupt many links to prevent messages from being delivered. The protocol, which we call DISPERSE, is just a "two-phase echo," and is described in Fig. 2.

We stress that the DISPERSE protocol does not guarantee that only authentic messages are received, nor does it ensure that messages are not retransmitted. The only guarantee it offers is that if there is a path in the network between N_i and N_j of length at most

Protocol DISPERSE(m, i, j)

Sending a string m from node N_i to node N_j.

1. Node N_i sends the message "`forward` m `to` N_j" to all other nodes.
2. Upon receipt of a message "`forward` m `to` N_j," allegedly from N_i, a node N_k sends the message "`forwarding` m `from` N_i" to N_j.
3. Node N_j marks all the strings m for which it received a message "`forwarding` m `from` N_i" as being received from N_i.

Fig. 2. Code of the DISPERSE protocol.

two, consisting only of reliable links, then any message that is sent between these nodes using the DISPERSE protocol arrives at its destination. In conjunction with our definition of operational nodes (Definition 5) we get the following lemma

Lemma 15. *Let n be the number of nodes in the network and let $s \leq \lfloor (n-1)/2 \rfloor$. If N_i, N_j are two s-operational nodes in some time interval, then during this time interval N_j receives every message that N_i sends to it using the* DISPERSE *protocol.*

Proof. According to Definition 5, if N_i, N_j are s-operational, then they both have reliable links to at least $n - s > n/2$ other s-operational (and, hence, nonbroken) nodes. Therefore there must exists at least one nonbroken node with reliable links to both N_i and N_j, and so this node will forward to N_j all the messages that N_i sends to it using the DISPERSE protocol. □

Authenticity. 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 either 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. Digital signatures require each node N_i to have the following:

(a) A pair of signature and verification keys for the centralized signature scheme CS. Below we respectively denote these keys by s_i^u and v_i^u where u is the time unit, and refer to them as the *local keys* of N_i in time unit u.
(b) A public verification key for the PDS scheme ALS, which we denote by $\mathsf{v}_{\mathsf{cert}}$ and refer to as the the *global verification key*.
(c) A signature on the assertion "`the public key of` N_i `in time unit` u `is` v_i^u" which can be verified with the global verification key. We call this signature the *certificate* of N_i during time unit u, and denote it by cert_i^u.

These keys are used in a standard way to authenticate messages. Namely, in order to authenticate a message m from node N_i to node N_j in communication round w during time-unit u, node N_i computes a signature on $\langle m, i, j, u, w \rangle$, and appends its local verification key and certificate. On the receiving end, N_j checks that the message has the right form, uses its global verification key $\mathsf{v}_{\mathsf{cert}}$ to verify the signatures on the certificate, and then verifies the signature on the message. More precisely, we have algorithms CERTIFY and VER-CERT that are described in Fig. 3, and protocol AUTH-SEND that is described in Fig. 4. In what follows we sometimes use the term *properly certified message* to describe a message that passes the verification algorithm.

Partial agreement. Our construction needs to have the nodes agree on the value to be signed. To that end we use the following standard PARTIAL-AGREEMENT protocol. Intuitively, the goal of this protocol is to ensure that there is a single value y such that every node either ends up with the value y, or ends up with no value at all. This protocol is described in Fig. 5. The property of this protocol needed for our construction is proved in Lemma 16.

Algorithm CERTIFY$\left(m, i, j, u, w, \mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{cert}_i^u\right)$

1. Compute a signature $\sigma \leftarrow C\mathsf{Sign}\left(\mathsf{v}_i^u, \mathsf{s}_i^u, \langle m, i, j, u, w\rangle\right)$.
2. Output $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}_i^u, \mathsf{cert}_i^u\rangle$.

Algorithm VER-CERT$(j, i, u, w, \mathtt{msg}, \mathsf{v}_{\mathsf{cert}})$

1. Check format: parse $\mathtt{msg} = \langle m, i', j', u', w', \sigma, \mathsf{v}, \mathsf{cert}\rangle$. If $i' \neq i$ (wrong source), $j' \neq j$ (wrong destination), or $u' \neq u$ or $w' \neq w$ (wrong time), then reject msg.
2. Check certificate: run

   $A\mathsf{Ver}(\text{“}\mathtt{the\ public\ key\ of}\ N_i\ \mathtt{in\ time\ unit}\ u\ \mathtt{is}\ \mathsf{v},\text{”}\,\mathsf{cert}, \mathsf{v}_{\mathsf{cert}})$.

   If verification fails, reject msg.
3. Check signature: run $C\mathsf{Ver}(\mathsf{v}, \langle m, i, j, u, w\rangle, \sigma)$. If verification fails, reject msg.
4. If all three steps succeed, *accept* msg.

Fig. 3. The CERTIFY and VER-CERT algorithms.

Lemma 16. *Assume that during some execution of protocol* PARTIAL-AGREEMENT *there exists a set S of nonbroken nodes of size at least* $\lceil (n+1)/2 \rceil$ *such that for each pair of nodes* $N_i, N_j \in S$:

(a) N_j *receives every message that* N_i *sends to it using the* DISPERSE *protocol.*
(b) N_j accepts *as authentic from* N_i *all and only the messages that* N_i *certifies.*

Then in this execution the following holds:

1. *If all the nodes in S start this execution with the same input value x, then they all output x.*
2. *If one of the nodes in S outputs a value* $y \neq \varphi$, *then every node in S outputs either the same value y or* φ. (*In other words, there exists a value y such that the output value of every node in S is in* $\{y, \varphi\}$.)

Protocol AUTH-SEND $\left(m, i, j, u, w, \mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{v}_{\mathsf{cert}}, \mathsf{cert}_i^u\right)$

Sending a string m from N_i to N_j in communication round w during time unit u using local keys $\mathsf{s}_i^u, \mathsf{v}_i^u$, global verification key $\mathsf{v}_{\mathsf{cert}}$ and certificate cert_i^u.

1. Node N_i sets $\mathtt{msg} = \text{CERTIFY}(m, i, j, u, w, \mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{cert}_i^u)$, and invokes protocol DISPERSE$(\mathtt{msg}, i, j)$.
2. Whenever protocol DISPERSE within node N_j marks a message msg as received from node N_i, node N_j invokes VER-CERT $(j, i, u, w, \mathtt{msg}, \mathsf{v}_{\mathsf{cert}})$ and *accepts m* from N_i if VER-CERT *accepts* it. (The value of w used in VER-CERT is exactly two communication rounds before the current one, which is supposed to be when the message was sent).

Fig. 4. Code of the AUTH-SEND protocol.

Protocol PARTIAL-AGREEMENT$(i, x, u, w, \mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{v}_{\mathsf{cert}}, \mathsf{cert}_i^u)$

Node N_i starts with input value x in communication round w of time-unit u and uses local keys $\mathsf{s}_i^u, \mathsf{v}_i^u$, global key $\mathsf{v}_{\mathsf{cert}}$, and certificate cert_i^u.

1. For every other node N_j, node N_i invokes AUTH-SEND($x, i, j, u, w, \mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{v}_{\mathsf{cert}}, \mathsf{cert}_i^u$).
2. N_i collects all the input values that were *accepted* in Step 1. Every node from whom N_i *accepts* messages in Step 1 with more than one input value is marked "cheater."

   If there exists a set MAJ_i of nodes that are not marked "cheaters," so that $|\mathsf{MAJ}_i| \geq \lceil (n+1)/2 \rceil$ and all the nodes in MAJ_i have the same input value, then we denote this value by y_i. If there is no such set then we denote $y_i = \varphi$, and $\mathsf{MAJ}_i = \emptyset$.
3. For any certified message $\mathtt{msg}$ that was *accepted* in Step 1 from a node in MAJ_i, node N_i invokes DISPERSE($\mathtt{msg}, i, j$) to distribute $\mathtt{msg}$ to all other nodes N_j.
4. N_i applies verification procedure VER-CERT($i, j, u, w, \mathtt{msg}, \mathsf{v}_{\mathsf{cert}}$) for each message $\mathtt{msg}$ that was received in Step 3 and that allegedly contains the input value of N_j. Every node from which N_i *accepts* messages in either Step 2 or 4 with more than one input value is marked "cheater."
5. Denote by MAJ_i' the set of nodes that were in MAJ_i in Step 2, and were not marked "cheaters" in Step 4. If MAJ_i' still contains at least $\lceil (n+1)/2 \rceil$ nodes, then y_i is the output value of N_i. Otherwise, N_i outputs φ.

Fig. 5. Code for protocol PARTIAL-AGREEMENT

Proof. 1. Since each node $N_i \in S$ only certifies a single input value, and the messages that other nodes in S *accept* from N_i in this execution E are those that N_i certifies, 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 is a majority value in all of them in Step 2, and, moreover, for each $N_i \in S$ we have $S \subseteq \mathsf{MAJ}_i$. This is still true in Step 4, and so $|\mathsf{MAJ}_i'| \geq |S| \geq \lceil (n+1)/2 \rceil$ for all $N_i \in S$, hence all the nodes in S output the value x.

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

However, N_k appears in MAJ_i' with input value y_i and in MAJ_j' with input value y_j. Therefore, in Step 3, 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 we assume that messages sent between N_i and N_j in the execution E is received, then both N_j and N_i must mark N_k as "cheater" in Step 4. Contradiction. □

4.2. *A PDS Scheme in the UL Model*

Fix n and t with $n > 2t$. Our construction of the PDS scheme $\mathsf{ULS} = (U\mathsf{Gen}, U\mathsf{Sign}, U\mathsf{Ver}, U\mathsf{Rfr})$ in the UL model, given the centralized signature scheme $\mathsf{CS} =$

($C\mathsf{Gen}, C\mathsf{Sign}, C\mathsf{Ver}$) and the PDS scheme $\mathsf{ALS} = (A\mathsf{Gen}, A\mathsf{Sign}, A\mathsf{Ver}, A\mathsf{Rfr})$, proceeds as follows.

Scheme ULS runs the scheme ALS when each message is sent via protocol AUTH-SEND using the centralized signature scheme CS. This guarantees existence of a large enough clique of nodes, among which the communication is authenticated and reliable. We show that such a clique is sufficient for the scheme ALS to remain secure within each time unit. At the beginning of each time unit we use the scheme ALS itself to certify each node's new keys for this time unit, as sketched at the beginning of Section 4. Below we describe the different components in detail.

4.2.1. *Key Generation, $U\mathsf{Gen}$*

Recall that since $U\mathsf{Gen}$ is executed during system set-up, we assume that no nodes are broken during its execution, and that all the messages arrive unmodified at their destination.

On security parameter k node N_i first executes $A\mathsf{Gen}$, the key generation protocol of ALS, to obtain the public key $\mathsf{v}_{\mathsf{cert}}$ and the secret share sh_i^0. Next N_i executes the key generation algorithm of CS to obtain $\langle \mathsf{v}_i^0, \mathsf{s}_i^0 \rangle \leftarrow C\mathsf{Gen}(k)$. Finally, N_i sends v_i^0 to all the nodes, and they all execute the distributed signature protocol $A\mathsf{Sign}$ to generate a certificate for v_i^0, namely,

$$\mathsf{cert}_i^0 \leftarrow A\mathsf{Sign}_{\mathsf{sh}_1^0,\ldots,\mathsf{sh}_n^0}(\text{“}\texttt{the public key of } N_i \texttt{ in time unit 0 is } \mathsf{v}_i^0\text{”})$$

Node N_i then "burns" $\mathsf{v}_{\mathsf{cert}}$ in its read-only memory, and stores $(\mathsf{sh}_i^0, \mathsf{v}_i^0, \mathsf{s}_i^0, \mathsf{cert}_i^0)$ in regular memory.

Throughout the protocol, during time unit u each node N_i has local keys $\mathsf{v}_i^u, \mathsf{s}_i^u$, share sh_i^u and certificate cert_i^u stored in regular memory.

4.2.2. *Signature and Verification, $U\mathsf{Sign}$, $U\mathsf{Ver}$*

The protocol $U\mathsf{Sign}$ is similar to the signature protocol $A\mathsf{Sign}$ of ALS, except that for every message m that is sent in $A\mathsf{Sign}$ from N_i to N_j in time unit u, the sender in $U\mathsf{Sign}$ invokes AUTH-SEND, using its current keys $\mathsf{s}_i^u, \mathsf{v}_i^u, \mathsf{cert}_i^u$, to send m. In addition, the receiver accepts its messages via AUTH-SEND.

The centralized verification algorithm remains unchanged, $U\mathsf{Ver} = A\mathsf{Ver}$.

4.2.3. *Refreshment Protocol, $U\mathsf{Rfr}$*

The refreshment protocol $U\mathsf{Rfr}$ is divided into two parts: In the first part each node generates new local keys of the centralized scheme CS and obtains a certificate for its new verification key. In the second part the nodes execute the refreshment protocol $A\mathsf{Rfr}$ of the PDS scheme ALS, with the exception that messages are transmitted via AUTH-SEND.

Part (I). This part can be thought of as taking place "at the end of the previous time unit" (i.e., time unit $u - 1$), since during this part nodes still use their local keys from

the previous time unit for authentication. Yet, we remind the reader that, technically speaking, the refreshment phase belongs to both time units.

1. Each node N_i runs the key generation algorithm of CS to obtain a new pair of local signature and verification keys $(\mathsf{s}_i^u, \mathsf{v}_i^u) \leftarrow C\mathsf{Gen}(k)$.
2. Next, N_i sends to all other nodes the message "`the public key of` N_i `in time unit` u `is` v_i^u." This message is sent "in the clear" without any authentication. (The reason is that N_i may be trying to recover from a break-in, and thus it may not have the necessary keys to authenticate its communication.)
3. When a node N_j receives a key v, allegedly from some other node N_i, then N_j invokes a copy of PARTIAL-AGREEMENT with v as the input, and using the keys from the time unit $u-1$, to get

$$\mathsf{v}_{i,j}^u = \text{PARTIAL-AGREEMENT}\left(j, \mathsf{v}, u-1, w, \mathsf{s}_j^{u-1}, \mathsf{v}_j^{u-1}, \mathsf{v}_{\mathsf{cert}}, \mathsf{cert}_j^{u-1}\right).$$

   Notice that in this step N_j participates in n copies of PARTIAL-AGREEMENT, one for each node N_i. These copies can all be executed in parallel. If N_j receives more than one value v allegedly from N_i, then N_j runs PARTIAL-AGREEMENT on the first such value. (It is stressed that PARTIAL-AGREEMENT is run only *once* per node per refreshment phase.)
4. If the output of N_j from the PARTIAL-AGREEMENT protocol from Step 3, $\mathsf{v}_{i,j}^u$, is different than φ, then N_j invokes the signature protocol $U\mathsf{Sign}$ (still using the keys from time-unit $u-1$) to obtain a signature on the assertion "`the public key of` N_i `in time unit` u `is` $\mathsf{v}_{i,j}^u$." If at the conclusion of the signature protocol N_j obtains a valid signature on that assertion, then it sends this signature to N_i.
5. If N_i receives from any node a valid signature with respect to $\mathsf{v}_{\mathsf{cert}}$ on the assertion "`the public key of` N_i `in time unit` u `is` v_i^u," then this signature becomes cert_i^u. Otherwise, N_i sets $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$. In this case N_i outputs "`alert`."

When Part (I) is over, each node N_i replaces the local keys s_i^{u-1}, v_i^{u-1}, and cert_i^{u-1} with s_i^u, v_i^u, and cert_i^u, respectively.

Part (II). This part can be thought of as happening "at the beginning of the current time unit" (i.e., time unit u) since nodes now use their new keys from Part (I) for authentication. Yet, the previous time unit is not done until the old secret keys of scheme ALS are erased.

In this part the nodes execute the share refreshment protocol $A\mathsf{RFR}$. As in the signature protocol, for every message m that is sent from N_i to N_j, the sender in $U\mathsf{RFR}$ invokes AUTH-SEND, using its new keys s_i^u, v_i^u, cert_i^u, to send m.

We stress that protocol $A\mathsf{RFR}$ instructs each node N_i to erase from its memory the share of the global signing key sh_i^{u-1}. The local output of this protocol becomes sh_i^u.

If, at the end of the refreshment protocol, a node N_i either has $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$ or has failed to refresh its share during Part (II), then N_i outputs "`alert`."

4.3. *Proof of Theorem* 14

Before presenting the formal analysis, we give a high-level overview. According to Definition 12, to prove that ULS is (t, t)-secure in the UL model, we need to show that for every (t, t)-limited UL-forger that interacts with ULS there exists an ideal-model forger such that the corresponding global outputs are indistinguishably distributed. Using our assumption that ALS is t-secure in the AL model, it is sufficient to show that for every (t, t)-limited UL-forger that interacts with ULS there exist a t-limited AL-forger that interacts with ALS and generates an indistinguishably distributed global output. More precisely, fix an arbitrary (t, t)-limited UL-forger $\mathcal{UF}$. It suffices to show a t-limited AL-forger $\mathcal{AF}$ interacting with ALS that satisfies

$$\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}} \overset{c}{\approx} \text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}. \tag{4}$$

In the rest of the proof we construct forger $\mathcal{AF}$ and show (4). In fact, the *statistical distance* between the two sides of (4) is shown to be negligible.

As usual, $\mathcal{AF}$ operates via a simulation of $\mathcal{UF}$, where the activity of the "upper layer" of ULS (which is identical to ALS) is carried out by the parties in the AL model, and the "lower layer" of ULS is imitated by $\mathcal{AF}$ itself. To prove (4) we define "good executions" as those where no messages are forged by the simulated adversary $\mathcal{UF}$, and where all the operational nodes have valid certificates for their local keys (Definition 18). Then we show that good executions have the same probability weight according to both $\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}$ and $\text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}$, and that executions which are not good have only negligible probability weight. The first assertion follows almost immediately from the way we define the simulator $\mathcal{AF}$ (Lemma 21). The second part is proven via a sequence of claims, roughly sketched below: Consider a partial execution E, which is good up to the end of time unit $u - 1$. Then:

- If N_i is operational in time unit u, then from the properties of PARTIAL-AGREEMENT it follows that many operational nodes participate in signing N_i's local public key in the refreshment phase at the beginning of this time unit (Lemma 20).
- It follows from the security of the ALS scheme in the AL model, combined with the fact that E does not contain any forged messages during the refreshment phase, that N_i will get a valid certificate on its local public key in this refreshment phase, except with negligible probability (Lemma 26).
- Similarly, since each operational node participates in signing at most one certificate for every node, it follows that if N_i obtains a certificate on its local public key, then $\mathcal{UF}$ *does not* obtain a certificate for N_i on any other key, except with negligible probability (Lemma 27).
- Finally, since the only certificate for N_i in this time unit is for the key v_i^u, then the only way to forge a message from N_i would be to sign it with respect to v_i^u. However, as long as N_i is not broken, $\mathcal{UF}$ does not know the corresponding secret key s_i^u and hence cannot sign messages with respect to v_i^u, except with negligible probability (Lemma 28).

We conclude that E is also a good execution up to the end of time unit u, except with negligible probability. (Note that, although the above sketch suggests a proof by induction, the actual proof proceeds by using $\mathcal{AF}$ to construct forgers for schemes ALS

and CS, that succeed in breaking their corresponding schemes with probability that is related to the probability of bad executions. This, in particular, allows comparing the relative security of schemes in the two models.)

The construction of UF is presented in Section 4.3.1, and analyzed in Section 4.3.2.

4.3.1. *The AL-Forger, $\mathcal{AF}$*

Let $\mathcal{UF}$ be any (t, t)-limited UL-forger. We now describe a t-limited AL-forger $\mathcal{AF}$ that simulates $\mathcal{UF}$. On a high level, $\mathcal{AF}$ runs the algorithm $\mathcal{UF}$ as a black-box, imitating the UL-model network that $\mathcal{UF}$ expects to interact with. To carry out this imitation, $\mathcal{AF}$ uses its access to the scheme ALS in the AL model, and implements by itself the parts in ULS that are not present in ALS (i.e., the "lower layer" of the two layers that constitute ULS). The operation of $\mathcal{AF}$ is demonstrated pictorially in Fig. 6.

Notations. In the description below we refer to the AL-network, which $\mathcal{AF}$ has access to, as the "real network" and the UL-network that $\mathcal{AF}$ imitates for $\mathcal{UF}$ we refer to as the "imitated network." Similarly, we distinguish between "real communication rounds" in the real network and "imitated communication rounds" in the imitated network (typically we need several rounds of the imitated network for every round in the real network). Also, the nodes in the real network are called "real nodes" and are denoted $N'_1, N'_2, \ldots$ and those in the imitated network are called "imitated nodes" and are denoted $N_1, N_2, \ldots$.

Initialization. When $\mathcal{AF}$ is started, it is given the security parameter k and the number of nodes n, its input x_F, a public key v_{cert} (generated in $A\mathsf{Gen}(k)$), and access to the

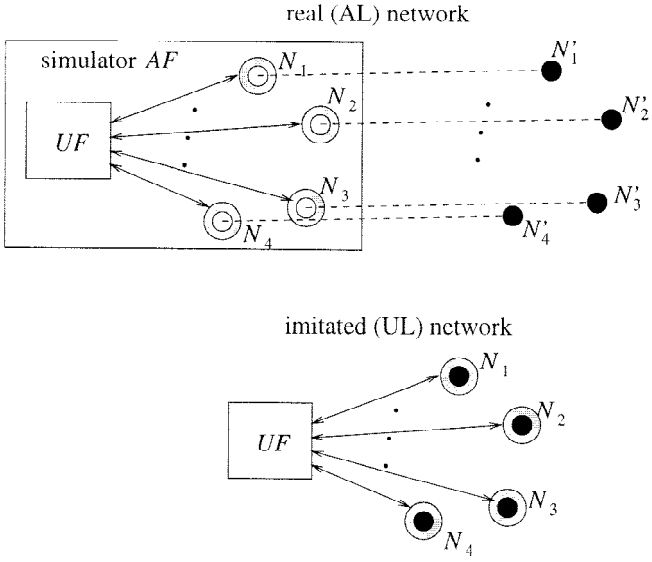

Fig. 6. Operation of the simulator $\mathcal{AF}$. The solid areas represent the upper layer of ULS (i.e., the scheme ALS). The grey areas represent the lower layer of ULS.

real network where n nodes are running the protocol ALS with shares of the secret key corresponding to $\mathsf{v}_{\mathsf{cert}}$.

$\mathcal{AF}$ then generates the following keys: For each imitated node N_i it runs a copy of the key-generation algorithm of CS to obtain $\langle \mathsf{v}_i^0, \mathsf{s}_i^0 \rangle \leftarrow CGen(k)$. Then for every i it asks all the nodes in the real network for a signature on the assertion "`the public key of` N_i `in time unit 0 is` v_i^0," and denotes the resulting signature by cert_i^0. Such signatures will be generated since the AL-model nodes receive the same message to sign and none of them is broken. Finally, $\mathcal{AF}$ initializes $\mathcal{UF}$, giving it k, n, x_{F}, and $\mathsf{v}_{\mathsf{cert}}$. Throughout the execution, during time unit u $\mathcal{AF}$ maintains for each imitated node N_i the local keys v_i^u, s_i^u and cert_i^u.

Execution. During the execution, $\mathcal{AF}$ interacts with the real nodes and the black-box representing $\mathcal{UF}$. In this interaction $\mathcal{AF}$ keeps track of the reliable links in the imitated network, by recording all the messages that it sends to and receives from $\mathcal{UF}$. This way, $\mathcal{AF}$ can also keep track of which imitated node is t-operational at any given time.

Below we describe the events that occur in the interaction of $\mathcal{AF}$ with $\mathcal{UF}$ and with the real nodes, and how $\mathcal{AF}$ reacts to these events.

Disconnected and broken imitated nodes. For each imitated node N_i, $\mathcal{AF}$ keeps a status variable, which can be either "operational" (if N_i is t-operational in the current imitated communication round), "disconnected" (if N_i is not t-operational but $\mathcal{UF}$ did not ask to break it), or "broken" (if $\mathcal{UF}$ asked to break it and did not ask to leave it yet).

When an imitated node N_i moves from "operational" to "disconnected" $\mathcal{AF}$ breaks into the corresponding real node N_i'. As long as N_i remains "disconnected" $\mathcal{AF}$ lets the broken real node N_i' execute the original protocol ALS, but it may occasionally change its memory contents to reflect acceptance of messages which were not actually sent in the real network, and it may also send messages on behalf of N_i to other nodes in the real network. (This reflects the intuition that since N_i is disconnected, it may appear to be broken in the eyes of the other nodes even if it is not really broken.)

If $\mathcal{UF}$ asks to break an imitated node N_i, $\mathcal{AF}$ marks N_i as "broken." Then it breaks into the real node N_i' (if it is not broken already) and provides $\mathcal{UF}$ with the contents of N_i''s memory (including N_i's input), and also with the keys v_i^u, s_i^u, cert_i^u that it maintains for the imitated N_i. When $\mathcal{UF}$ asks to change the contents of the memory of a broken imitated node N_i, $\mathcal{AF}$ makes the corresponding changes in the memory of the real node N_i' or the keys that it maintains for N_i.

When $\mathcal{UF}$ leaves an imitated node N_i, then $\mathcal{AF}$ changes the status of N_i from "broken" to "disconnected," *but does not leave the real node* N_i' *yet*. $\mathcal{AF}$ leaves N_i' only when the imitated node N_i becomes "operational."

Real communication rounds. At the beginning of every real communication round, $\mathcal{AF}$ records all the messages that were sent by the nonbroken nodes in the real network. For every message m that the nonbroken real node N_i' sends to real node N_j', $\mathcal{AF}$ imitates an execution of the protocol AUTH-SEND, among the imitated nodes, using the local keys v_i^u, s_i^u, cert_i^u that it maintains for the imitated N_i, and the global verification key $\mathsf{v}_{\mathsf{cert}}$.

Notice that the imitation of this protocol does not involve any communication in the real network. Rather, it consists only of messages sent back and forth between $\mathcal{AF}$ and its black-box $\mathcal{UF}$, where $\mathcal{AF}$ executes the parts of all the imitated nodes that are not marked as "broken."

If an imitated node N_j, which is marked "disconnected," *accepts* a message string m from another imitated node N_i, then $\mathcal{AF}$ modifies the memory of the real node N'_i (which is broken) to reflect acceptance of the message m from the real node N'_i. Also, if an imitated node N_j *accepts* a message string m from another imitated node N_i that is marked "disconnected," then $\mathcal{AF}$ sends m to the real node N'_j on behalf of the (broken) real node N'_i.

Bad event (A): If an imitated node N_j that is marked "operational" *accepts* a message string m from another imitated node N_i that is also marked "operational," and the real node N'_i did not send the message m to the real node N'_j in this communication round, then $\mathcal{AF}$ outputs "`failure`" and halts.

Imitated global signatures. When imitated node N_i that is not "broken" gets a request from $\mathcal{UF}$ to sign message m, then $\mathcal{AF}$ asks the real node N'_i to sign message m. The real nodes then execute the protocol $A\mathsf{Sign}$, with $\mathcal{AF}$ treating the real communication rounds as described above.

Imitated refreshment phases. Below we require that the imitated refreshment phases begin before the refreshment phases in the real network, so that Part (II) of the refreshment protocol $U\mathsf{Rfr}$ is aligned with the refreshment protocol $A\mathsf{Rfr}$. (This means that the time units in the imitated network start slightly before the time units in the real network.) At the beginning of an imitated refreshment phase in time-unit u, $\mathcal{AF}$ imitates an execution of Part (I) of the refreshment phase $U\mathsf{Rfr}$. This is done by executing the following procedure for every imitated node N_i that is not marked "broken":

1. Execute a copy of the key-generation algorithm of CS to obtain $\langle \mathsf{v}_i^u, \mathsf{s}_i^u \rangle \leftarrow C\mathsf{Gen}(k)$.
2. Send v_i^u to all the imitated nodes. This amounts to giving $\mathcal{UF}$ the message v_i^u on behalf of the imitated node N_i, and asking that this message be delivered to all the nodes.
3. For each imitated node N_j that is not marked "broken," invoke a copy of protocol PARTIAL-AGREEMENT, using as input value the string that $\mathcal{UF}$ delivered on behalf of N_i (of course, this string may or may not be equal to v_i^u).

   We note again that this imitated execution does not involve any communication in the real network, but only involves messages sent back and forth between $\mathcal{AF}$ and its black-box $\mathcal{UF}$.

   Bad event (B): If at the conclusion of an imitated invocation of PARTIAL-AGREEMENT there exist two imitated "operational" nodes with two different local output values, both different from φ, then $\mathcal{AF}$ outputs "`failure`" and halts.
4. For every imitated node N_j that is not "broken," if the local output of N_j from the execution of PARTIAL-AGREEMENT was $y_j \neq \varphi$, then $\mathcal{AF}$ asks the real node N'_j for a signature on the assertion "`the public key of` N_i `in time unit` u `is` y_j." The real nodes then execute the protocol $A\mathsf{Sign}$, with $\mathcal{AF}$ treating the real communication rounds as described above.

5. If at the conclusion of the signature protocol, real node N'_j returns a valid signature on the value y_j, then $\mathcal{AF}$ asks $\mathcal{UF}$ to deliver this value to imitated node N_i on behalf of imitated node N_j.
6. If $\mathcal{UF}$ delivers to imitated node N_i a valid signature on the assertion "`the public key of` N_i `in time unit` u `is` v_i^u," then $\mathcal{AF}$ sets this signature to be cert_i^u. Otherwise, $\mathcal{AF}$ sets $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$.

   Recall that in this case a node in the UL model would output `alert`. To emulate this behavior in the AL model, $\mathcal{AF}$ instructs the (corrupted) N_i in the AL model to output `alert.` Here we use in an essential way the fact that nodes output `alert` only when they are not t-operational.

When this process is over, $\mathcal{AF}$ imitates Part (II) of the refreshment phase by invoking the refreshment protocol $A\mathsf{RFR}$, treating the real communication rounds as described above.

Bad event (C): If at the end of the refreshment phase, there exists an imitated "operational" node N_i with $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$, then $\mathcal{AF}$ outputs "`failure`" and halts.

Output. When $\mathcal{UF}$ halts with some output, $\mathcal{AF}$ outputs whatever $\mathcal{UF}$ does and halts. If any of the bad events (A), (B) or (C) happens, then the output of $\mathcal{AF}$ is the transcript of the execution up to this point, followed by the word "`failure`."

4.3.2. *Analyzing* $\mathcal{AF}$

We start by noting that $\mathcal{AF}$ is a "legal" AL-model adversary since it never modifies messages sent over the links of the real network. Also, at any time during the simulation the number of nodes that $\mathcal{AF}$ breaks equals the number of nodes that are either broken or t-disconnected in the imitated execution of the UL-model network with $\mathcal{UF}$. Hence, if $\mathcal{UF}$ is a (t, t)-limited UL-forger, then $\mathcal{AF}$ is a t-limited AL-forger.

We proceed to show that $\mathcal{AF}$ simulates $\mathcal{UF}$ with only negligible deviation. That is,

$$\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}} \overset{s}{\approx} \text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}. \tag{5}$$

(Recall that $\overset{s}{\approx}$ negligible *statistical* distance.) This is shown as follows. We first observe that as long as none of the bad events (A), (B), or (C) happens, the output of the AL-model forger $\mathcal{AF}$ described above is also a syntactically valid output for the UL-model forger $\mathcal{UF}$. More formally, fix a value of the security parameter k and an input vector $\vec{x}$ for the rest of the proof. Then any execution in the support set of $\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x})$ in which the bad events (A), (B), and (C) do not happen is also in the support set of $\text{UL-TRANS}_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x})$. (Here we identify an execution and its transcript.)

Identify a set GOOD of "good executions." Next we demonstrate the following two facts:

Fact 1. Conditioned on the set GOOD, the above two distributions (on the global outputs of executions) are identical. In fact, we show that in this case even the distributions on transcripts are identical. Namely, for any execution $E \in \mathsf{GOOD}$ we have

$$\Pr\left[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}) = E\right] = \Pr\left[\text{UL-TRANS}_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x}) = E\right].$$

Fact 2. The probability weight of executions that are not good is negligible in both distributions. That is, there exists some negligible function $\nu(k)$ such that

$$\Pr\left[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \notin \mathsf{GOOD}\right] < \nu(k)$$

and

$$\Pr\left[\text{UL-TRANS}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x}) \notin \mathsf{GOOD}\right] < \nu(k).$$

The combination of Facts 1 and 2 proves (5) (and so completes the proof of the theorem). We start by defining good executions.

Definition 17 (Forged Messages). Let E be an execution of ULS with $\mathcal{UF}$ in the UL model, and let $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$ be a message, delivered by $\mathcal{UF}$ to some node in the network allegedly from node N_i during time unit u within the execution E (but outside Part (I) of the refreshment phase at the beginning of this time unit, since messages that were sent during this part of the refreshment protocol are verified with the keys of the time unit $u-1$, and are considered in that time unit).

We say that msg is *forged* if:

(a) msg is properly certified, i.e., $\text{VER-CERT}(j, i, u, w, \mathtt{msg}, \mathsf{v}_{\mathsf{cert}}) = \textit{accept}$, where $\mathsf{v}_{\mathsf{cert}}$ is the global verification key in this execution.

(b) N_i did not send any properly certified message $\mathtt{msg}' = \langle m, i, j, u, w, \sigma', \mathsf{v}_i^u, \mathsf{cert}_i^u\rangle$, with the same values of m, i, j, u, w as in msg, during communication round w in time unit u.

(Intuitively, we mean to say that N_i did not send msg in that round. However, since there could be many different valid signatures on the same message and it may be possible to obtain one valid signature from another one on the same message, we have to use the above complicated-looking formal condition.)

(c) N_i was not broken in time unit u, up to and including communication round w. Moreover, at the end of Part (I) of the refreshment phase in time unit u, N_i has s_i^u, v_i^u, and cert_i^u different than φ. (Notice that, in our scheme, $\mathsf{cert}_i^u \neq \varphi$ implies that cert_i^u is a valid certificate for v_i^u.)

Definition 18 (Good Executions). Let E be an execution of $\mathcal{UF}$ with the UL-model network. We say that E is a *good execution* up to time unit u, if

- E does not contain forged messages until the end of time unit u, and
- for every node N_i that is t-operational in any time unit $u' \leq u$, it holds that $\mathsf{s}_i^{u'}$, $\mathsf{v}_i^{u'}$, and $\mathsf{cert}_i^{u'}$ are all different than φ.

Let GOOD be the set of executions that are good throughout.

Note that each execution in GOOD belongs to the support set of both $\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x})$ and $\text{UL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x})$. Also, there is an efficient algorithm which, given the transcript of an execution, decides whether this execution is good.

Before we proceed with proving the two above facts, we need to prove two technical lemmas about the executions of protocol PARTIAL-AGREEMENT in the refreshment phases.

Lemma 19. *Let E be an execution of* ULS*, which is good up to time unit $u-1$, let E' be any execution of* PARTIAL-AGREEMENT *in E up to the refreshment phases at the beginning of time unit u, and denote by S the set of nodes that are t-operational throughout the execution E'. Then, at the end of the execution E', there exists a single value y such that the local output of each node in S is either y or φ.*

Proof. It is enough to show that the set S satisfies the premise of Lemma 16. First, since $\mathcal{UF}$ is (t,t)-limited and $n \geq 2t+1$, then the size of S is at least $|S| \geq n-t \geq \lceil (n+1)/2 \rceil$. Next, since all the nodes in S are t-operational and since every message is sent using the DISPERSE protocol, then by Lemma 15 every node in S receives every message that every other node in S sends to it.

Since E does not contain forged messages up to time unit $u-1$ (which includes the refreshment phase at the beginning of time unit u), then the only properly certified messages that nodes in S receive from other nodes in S, and hence the only ones they *accept*, are the ones that these other nodes sent. Finally, since E is good then all the nodes in S have valid local keys and certificates for round $u-1$, and so every message that is sent by a node in S is properly certified, and thus is *accepted* by all nodes in S. □

Lemma 20. *Let E be an execution of* ULS *which is good up to time unit $u-1$, and let N_i be a node that is t-operational at the end of the refreshment phase of time unit u in E. Then there exists a set S of at least $n-t$ nodes such that*:

(a) *All the nodes in S are operational throughout this refreshment phase.*
(b) *At the conclusion of the invocation of* PARTIAL-AGREEMENT *regarding N_i's local public key, the local output of all the nodes in S is equal to v_i^u, the public key that N_i sent to them in this refreshment phase.*

Proof. Recall from Definition 5 that since N_i is operational at the end of the refreshment phase at the beginning of time unit u, then it has reliable links to a set S of at least $n-t$ nodes that are operational throughout this refreshment phase. Using the same arguments as in Lemma 19, this set S satisfies the premise of Lemma 16. Also, since N_i had reliable links to all of them, then they all start PARTIAL-AGREEMENT with v_i^u as their local input. Now Lemma 16 implies that they all end the execution of PARTIAL-AGREEMENT with v_i^u as their local output. □

Lemma 21 (Fact 1). *For any execution $E \in$ GOOD we have*

$$\Pr\left[\text{AL-TRANS}_{\text{ALS},\mathcal{AF}}(k,\vec{x}) = E\right] = \Pr\left[\text{UL-TRANS}_{\text{ULS},\mathcal{UF}}(k,\vec{x}) = E\right].$$

Proof. We show that there exists a bijection sim: $\{0,1\}^* \to \{0,1\}^*$ between the randomness used in $\text{AL-TRANS}_{\text{ALS},\mathcal{AF}}(k,\vec{x})$ and the randomness used in $\text{UL-TRANS}_{\text{ULS},\mathcal{UF}}(k,\vec{x})$, so that for every security parameter k and randomness $\vec{r}$:

1. If $\text{AL-TRANS}_{\text{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) \notin$ GOOD, then also $\text{UL-TRANS}_{\text{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r})) \notin$ GOOD.
2. If $\text{AL-TRANS}_{\text{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) \in$ GOOD, then $\text{UL-TRANS}_{\text{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r})) \in$ GOOD.

We construct $sim(\vec{r})$ as follows. Intuitively, $sim(\vec{r})$ is the random-input vector that corresponds to the execution of the imitated network within $\mathcal{AF}$, in the execution AL-TRANS$_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}, \vec{r})$. More precisely, $sim()$ takes the part of $\vec{r}$ used by $\mathcal{AF}$ to imitate P_i for $\mathcal{UF}$, and moves it to the random input of P_i itself. That is, let $\vec{r} = r_0 \dots r_n$ be a random-input vector for an execution of ALS with $\mathcal{AF}$, and let $r_{0,i}$ be the part of r_0 that is used by $\mathcal{AF}$ to imitate P_i in the imitated network. Then in $sim(\vec{r})$ the value $r_{0,i}$ is removed from r_0 and is added to r_i at the appropriate location.

It is clear that $sim()$ is one-to-one and onto (since it only reorders the bits of its input). Moreover, from the construction we also have that if AL-TRANS$_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}, \vec{r})$ is not a good execution, then neither is UL-TRANS$_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x}, sim(\vec{r}))$.

We show that AL-TRANS$_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}, \vec{r}) \in$ GOOD implies that

$$\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}, \vec{r}) \in \mathsf{GOOD}.$$

In particular, we show that in this case AL-TRANS$_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}, \vec{r})$ = UL-TRANS$_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x}, sim(\vec{r}))$. For that, we only need to show that $\mathcal{AF}$ never outputs "`failure`" in a good execution. Bad events (A) and (C) in the definition of $\mathcal{AF}$ are excluded by the definition of good executions, and bad event (B) is ruled out by Lemma 20. The lemma follows. □

It remains to show Fact 2. This is done as follows: We first define three sets of executions, BAD1, BAD2, and BAD3, corresponding to the three types of failures in ULS, such that each execution $E \notin$ GOOD must be in either BAD1, BAD2, or BAD3. Then we show that the probability of each one of these three sets is negligible in the security parameter k.

Definition 22. An execution E is said to be a *bad execution of the first type* if there exists a time unit u such that E is good up to time unit $u-1$, but at the end of the refreshment phase in time unit u there exists a t-operational node N_i for which $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$. The set of bad executions of the first type is denoted BAD1.

Definition 23. An execution E is said to be a *bad execution of the second type*, if E is not a bad execution of the first type, and there exists a time unit u such that:

(a) E is good up to time unit $u-1$.
(b) E contains a forged message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$.
(c) The first such forged message in E has $\mathsf{v} \neq \mathsf{v}_i^u$ (i.e., the local public key in `msg` is not the local public key for which N_i obtained a certificate in the refreshment protocol).

The set of bad executions of the second type is denoted BAD2.

Definition 24. An execution E is said to be a *bad execution of the third type* if it is not a bad execution of the first or second type, and there exists a time unit u such that:

(a) E is good up to time unit $u-1$.
(b) E contains a forged message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$.

(c) The first such forged message in E has $\mathsf{v} = \mathsf{v}_i^u$ (i.e., this is the local public key for which N_i obtained a certificate in the refreshment phase).

The set of bad executions of the third type is denoted BAD3.

Lemma 25. *Every execution in the support set of* $\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x})$ *is either in* BAD1, BAD2, BAD3 *or in* GOOD.

Proof. Since every execution is vacuously good up to time unit 0, then every nongood execution has a last time unit up to which it is good. Since all the ways that an execution can stop being good are covered in Definitions 22–24, then every nongood execution must be in either BAD1, BAD2, or BAD3. □

In Lemma 26 we show that bad executions of the first type correspond to failures of ALS where the nodes cannot generate a certificate for a "good local key." Since we assume that ALS is secure, these executions have only a negligible probability. Formally, we prove

Lemma 26. *Let* $\varepsilon_1(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}) \in \text{BAD1}]$. *Then* $\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x})$ *can be distinguished from the output of any ideal-model adversary with an advantage of at least* $\varepsilon_1(k)$.

Proof. Let $E \in$ BAD1, and consider the global output of E during the refreshment phase at the beginning of time unit u. From Lemma 19 it follows that in each execution of PARTIAL-AGREEMENT, at most one value $y \neq \varphi$ is output by t-operational nodes, and so bad event (B) does not happen.

Moreover, by Lemma 20, there is a set of at least $n-t$ t-operational nodes which all have local output v_i^u from PARTIAL-AGREEMENT. Therefore, $\mathcal{AF}$ will ask the corresponding $n-t$ real nodes for a signature on v_i^u, and so all these real nodes will output this request to sign v_i^u. However, we know that none of the corresponding real nodes can obtain a signature on v_i^u. (Otherwise, $\mathcal{AF}$ would have the corresponding imitated nodes send this signature to N_i over the reliable links that N_i has with them, and then N_i would not have $\mathsf{s}_i^u = \mathsf{v}_i^u = \mathsf{cert}_i^u = \varphi$).

It follows that in any execution $E \in$ BAD1 there must exist a set of at least $n-t$ real nodes, which log a request for a signature on v_i^u but do not log a message confirming that it was indeed signed. Such an execution cannot happen in the ideal model, regardless of what the ideal-model adversary does. □

Next we show that bad executions of the second type correspond to a failure of ALS where the adversary can generate a certificate for a "bad local key." Again, since ALS is secure, then these executions too have only a negligible probability.

Lemma 27. *Let* $\varepsilon_2(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}) \in \text{BAD2}]$. *Then there exists a* t*-limited AL forger* $\mathcal{AF}'$ *such that* $\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}'}(k, \vec{x})$ *can be distinguished from the output of any ideal-model adversary with an advantage of at least* $\varepsilon_2(k)$.

Proof. To capture the intuition that $\mathcal{UF}$ can only get a certificate for the "wrong key" by forging signatures of the underlying AL-model PDS, we make use of the signature verifier V of the PDS scheme. Recall that the signature verifier is an external, unbreakable node, running only the verification algorithm of the PDS. When the scheme is started with the global verification key $\mathsf{v}_{\mathsf{cert}}$, V can be invoked on pairs (x, σ) and as a result it outputs "x `is verified`" if σ is a valid signature on x with respect to $\mathsf{v}_{\mathsf{cert}}$, and outputs nothing otherwise. The output of V then becomes part of the global output of the protocol (and in particular it needs to be simulated in the ideal model for the protocol to be secure).

$\mathcal{AF}'$ behaves exactly like $\mathcal{AF}$, but in addition it also queries V on some pairs. Specifically, for every message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$, delivered by $\mathcal{UF}$ to any imitated node, $\mathcal{AF}'$ invokes V to verify the signature cert on the message "`the public key of` N_i `in time unit` u `is` v."

Let $E \in \mathsf{BAD2}$, and let $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$ be the first forged message in E. Using the same argument as in the proof of Lemma 26 above, there is a set of at least $n-t$ t-operational nodes, all with local output $\mathsf{v}_i^u \neq \varphi$ from the corresponding invocation of PARTIAL-AGREEMENT. It now follows from Lemma 19 that no t-operational node has any other local output, and in particular no t-operational node has the local output v (which appears in the forged message msg).

Therefore, there cannot be more than t real nodes that are asked to sign the assertion $x =$ "`the public key of` N_i `in time unit` u `is` v." On the other hand, since msg is forged then cert is a valid signature on the above assertion, and hence V will output "x `is verified`."

We conclude that any bad execution of the second type contains a message x for which at most t nodes output a request for signature, and yet the signature verifier confirms that it is signed, and such an execution cannot occur in the ideal model. □

In the next lemma we show that bad executions of the third type correspond to a failure of the centralized signature scheme CS. Again, since CS is secure, it follows that they too occur only with a negligible probability.

Lemma 28. *Let* $\varepsilon_3(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}) \in \mathsf{BAD3}]$, *and let* $p(k)$ *be a polynomial upper bound on the number of time units in the executions of* ALS *with security parameter* k. *Then there exists a forger* $\mathcal{CF}$ *for the centralized signature scheme* CS, *which obtains existential forgery using an adaptive chosen message attack, with probability of at least* $\varepsilon_3(k)/(n \cdot p(k))$.

Proof. $\mathcal{CF}$ is given a public key v for the scheme CS and access to a signer with the corresponding signing key, and its goal is to achieve existential forgery under an adaptive chosen message attack.

$\mathcal{CF}$ first picks uniformly at random a time unit $u \in \{1, \ldots, p(k)\}$ and a node $i \in \{1, \ldots, n\}$. It then imitates an entire execution of $\mathcal{AF}$, playing the role of all the real nodes, $\mathcal{AF}$ itself and $\mathcal{UF}$. The only difference is that in time unit u, instead of picking the local keys for imitated node N_i using the key-generation algorithm, it sets the public key $\mathsf{v}_i^u = \mathsf{v}$ (where v is the public key that $\mathcal{CF}$ got as input). When node N_i needs to sign a

message relative to v_i^u, $\mathcal{CF}$ uses its access to the signer to get this signature. Finally, if this execution turns out to contain a forged message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, \mathsf{v}, \mathsf{cert}\rangle$ (with i, u, and v as above), then $\mathcal{CF}$ outputs the message/signature pair $(\langle m, i, j, u, w\rangle, \sigma)$ which is a valid pair with respect to v, and $\mathcal{CF}$ did not ask for a signature on $\mathtt{msg}$. In this case we say that the forgery was successful.

It is straightforward to see that the simulated $\mathcal{AF}$ sees exactly the same view as in a real interaction with scheme ALS, and that successful forgery occurs with probability at least $\varepsilon_3(k)/(n \cdot p(k))$. □

Lemmas 26–28 yield

Corollary 29 (Fact 2). *Bad execution of the first, second, and third type have a negligible probability weight in* $\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x})$. *Namely,*

$$\Pr\left[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k, \vec{x}) \notin \mathsf{GOOD}\right] = \Pr\left[\text{UL-TRANS}_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x}) \notin \mathsf{GOOD}\right] < \nu(k),$$

where $\nu(\cdot)$ *is a negligible function.*

This concludes the proof of Theorem 14. □

4.3.3. *Remarks on the Construction*

An extra property of the scheme ULS. Some components of the construction above are not strictly needed to obtain secure PDS schemes, but are important to ensure some extra properties, that are used to obtain awareness in the construction of authenticators in the next section. Clearly, the `alert` outputs are not used in the analysis of the scheme above, but are needed for awareness. Moreover, it can be verified that Step 3 of the refreshment phase (i.e., protocol PARTIAL-AGREEMENT) is not necessary for proving that ULS is (t, t)-secure. The purpose of this step is to ensure the following property:

Recall that in the security proof we proved that, for every (t, t)-limited UL adversary $\mathcal{UF}$ and security parameter k, there is a set GOOD of executions with probability weight $1 - \nu(k)$ under the distribution $\text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}(k, \vec{x})$ where $\nu(k)$ is negligible in k, and such that each execution in GOOD satisfies the following requirement: if N_i, N_j are not *broken* in time unit u, and if s_i^u, v_i^u, and cert_i^u are different than φ, then every message that N_j *accepts* from N_i was indeed sent by N_i in this time unit.

In the proof of Theorem 14, it suffices to apply this requirement only to nodes N_i that are *operational* during this time unit. Yet, the fact that this requirement applies even to nonoperational nodes is used in Section 5 to show *awareness* of the authenticator based on ULS for the case that the adversary is (t, t)-limited.

Stronger Adversaries. In the definition of reliable links (Definition 4) we require that messages will not be injected to these links. Yet, the scheme ULS remains secure even if we allow the adversary to inject messages on reliable links at all times, *except* during Step 2 in Part (I) of the refreshment phase. In other words, whenever the nodes have valid keys they can recognize and discard bogus messages. The only time where injecting bogus messages can be harmful is during the first round of the refreshment phase, where nodes are trying to get their new keys across to the other nodes, in an unverified way.

We also note that although the adversary can break the protocol by injecting too many bogus messages during that step, we would still get awareness in this case, since nodes will notice that they did not obtain certificates and will output "`alert`."

5. Emulating Authenticated Channels

In this short section we describe and prove the security of our proactive authenticator. As seen below, most of the work was done in Section 4, and very little remains to be said here.

A modular way to construct a proactive authenticator given a PDS scheme S in the UL model may proceed as follows. Given a protocol π (designed for the AL model), the nodes first run S; at each communication round w, each node N_i that is instructed by π to send a message m to node N_j asks all nodes to invoke the signing protocols of S on the extended message $\langle m, l, i, j \rangle$. Once a signature is generated, all the nodes will forward the extended message, accompanied by the generated signature, to N_j. In addition, it is specified that the nodes do not sign more than one message from N_i to N_j at each round, that N_j rejects unsigned messages, and that N_i outputs "`alert`" if it does not obtain the required signature.

The above approach, although modular, is very inefficient: it requires an invocation of the signing protocol of the PDS scheme S for every message. We avoid this inefficiency (at the price of breaking the modularity) by noticing that our construction of a PDS scheme in the UL model (Section 4.2) already provides the nodes with signing keys for a centralized signature scheme and with certificates for the corresponding verification keys. These can be used directly to authenticate the messages of the "higher layer" protocol π. That is, our authenticator, denoted Λ, is very similar to the PDS scheme ULS. It starts with a PDS scheme $\mathsf{ALS} = \langle A\mathsf{Gen}, A\mathsf{Sign}, A\mathsf{Ver}, A\mathsf{Rfr} \rangle$ *in the AL model* and a centralized signature scheme CS. Given a protocol π (designed for the AL model), Λ proceeds as follows:

1. Modify π by adding the key generation algorithm $A\mathsf{Gen}$ to the set-up phase of π, and executing the refreshment protocol $A\mathsf{Rfr}$ at the beginning of every time unit. The resulting protocol is denoted $\pi_{+\mathsf{ALS}}$.
2. Transform $\pi_{+\mathsf{ALS}}$ into a protocol for the UL model exactly as ALS is transformed to construct scheme ULS in Section 4. The messages of π are handled in the same way as the messages of $A\mathsf{Sign}$. That is, each message is sent (and received) using protocol AUTH-SEND.[1]
3. The outputs of π, as well as the "`alert`" outputs of ULS, are copied to the output of the constructed protocol. The other outputs of ULS are kept internal and are *not* copied to the output.

Let $\Lambda(\pi)$ denote the resulting protocol.

[1] Alternative constructions may simply send each π-message signed and certified (not via AUTH-SEND), or even exchange a secret key between each two parties and authenticate π-messages using that key. Such construction does not guarantee *delivery* of messages, thus they are not authenticators according to our definition; yet they provide authentication according to the standard interpretation of this term, namely, that only authentic messages are accepted as such.

Theorem 30. *Let $n \geq 2t + 1$. If the PDS scheme in use is t-secure in the AL model, then the authenticator Λ described above is t-emulating.*

Proof. The proof is almost identical to the proof of Theorem 14 (except for small formalities). □

Proposition 31. *Let $n \geq 2t + 1$. Then the authenticator Λ is (t, t)-aware.*

Proof. We show that, as long as the adversary is (t, t)-limited, *every* nonbroken node that is impersonated outputs "`alert`" in the same time unit. Recall that a node with $\mathsf{cert}_i^u = \varphi$ always outputs "`alert`." Furthermore, it follows from the extra property of scheme ULS (see the remark at the end of Section 4) that a nonbroken node for which $\mathsf{cert}_i^u \neq \varphi$ cannot be impersonated (except with a negligible probability). □

5.1. *Dealing with Stronger Adversaries*

We briefly discuss some properties of our scheme when the UL adversary is *not* (t, t)-limited. In most distributed settings, if the adversary can corrupt players beyond some preset bound, then "all bets are off," and the protocol cannot guarantee any security. In our setting, this problem is even more acute since we must also assume limitations on the ability of the adversary to tamper with communication links. As it is often easier to disrupt communication links than it is to break into nodes in the network, it is desirable to have a mechanism that can at least alert us to the fact that the adversary is disrupting too many links.

It is important to note the difference between this type of "awareness" and the type that we discussed above. The awareness condition from above is *local* (i.e., a node should be aware of its own condition), it is supposed to hold throughout the protocol, and it only holds as long as the adversary is appropriately limited. The awareness here, on the other hand, is *global* (i.e., we would like some node to figure out that something in the protocol went wrong) and it may only hold the first time something goes wrong (since after that we may not be able to guarantee anything anymore), but we would like it to hold even in the presence of stronger adversaries.

In the current solution we only partially guarantee this global awareness property: we can only achieve it when the adversary is capable of injecting too many bogus messages on the links, but otherwise is not stronger than in our assumptions. Specifically, we consider an adversary which is "almost (t, t)-limited," except that it can inject messages on arbitrarily many links. Call this type an almost (t, t)-limited adversary. In the presence of an almost (t, t)-limited adversary, we can no longer ensure t-emulation, since this adversary can send many bogus public keys in Step 2 of Part (I) of the refreshment phase (see Section 4.2.3), thereby preventing the nodes from obtaining certificates. However, if this happens, then these nodes output "`alert`." Now we have two cases: either this only happens to a few nodes, in which case the emulation can still go through, or it happens to "too many nodes," in which case we can detect that the adversary is not (t, t)-limited.

As of yet, we do not know how to guarantee the "global awareness" in case the adversary can also modify or delete messages on too many links. This drawback could be overcome if we had a secure PDS scheme in a model where the links are authenticated

but not necessarily reliable (i.e., where the adversary cannot inject or modify, but can discard arbitrarily many messages on the links). It is clear that in this case we cannot guarantee that the nodes be able to produce signatures, but it may still be possible to ensure that the adversary cannot forge signatures. However, we do not know whether such PDS schemes exist; this is an interesting open problem.

6. Discussion

This concluding section further discusses a few aspects of the definition and constructions.

The ROM assumption. Recall that our solution assumes the existence of ROM in each node, which the attacker can read but cannot modify. This requirement seems justifiable as it can be implemented in a variety of ways, and corresponds neatly to the well-accepted mechanisms for virus detection and removal. Note that we actually need two different types of ROM: The program can be stored in a memory that is "burned in" by the factory and never modified since. In contrast, the public key of the PDS scheme is written only during the start-up (or installation) phase, and varies from system to system.

One way to implement the ROM is by using a backup tape (or a CD-ROM) which is read during each refreshment phase. Another possibility is taking advantage of an appropriately designed operating system, which can guarantee a "virtual ROM" in software. That is, the OS can deny writing access to certain memory locations from *any* process, except processes that are active only at start-up time. This makes it easy to put a public key in this virtual ROM, hence facilitating our solution.

In the absence of ROM in the nodes, one can use the network itself as a source of "reliable memory," as described in [30]: At the beginning of each refreshment phase the nodes send each other the verification key of the PDS in use, and each node decides on the "right key to use" by majority vote. However, this solution has a few drawbacks. For one, the nodes still need some ROM to store the recovery code itself. In addition, this solution does not guarantee awareness: a node can be impersonated without knowing it, if the adversary sends to it the wrong key at the beginning of the refreshment phase.

Dealing with erasures. A basic assumption underlying the proactive approach is that the nodes successfully and completely *erase* certain pieces of sensitive data in each refreshment phase. (Specifically, in our protocol the nodes must erase their old shares of the signing key of the PDS scheme.) Here, if a node fails to erase this data, then the security of *all the nodes* may be compromised.

Note that successfully erasing data is not a trivial task. It is not enough to simply release the appropriate memory locations—the data has to be overwritten. (In fact, if the data was stored on magnetic media for more than a few seconds, then even overwriting it may not be sufficient.) Also, involuntary data back-ups (say, by the operating system) must be prevented. Therefore, trusting *other nodes* to erase data locally may be problematic in a completely distributed setting where nodes are not fully trusted.

Nonetheless, this assumption is necessary for any proactive solution, and there are many realistic settings in which it is a reasonable assumption to make. (For example,

notice that old secrets should only be "erased" to the point that they are not available to the adversary in case of a break-in. Hence, backup tapes that are kept off-line may be acceptable, if we only worry about "remote break-ins" from the network. See [11] and [10] for further discussion of this issue.

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 in these local neighborhoods, with each neighborhood running 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. In addition, there would also be a "higher level" PDS scheme, which can be run to regain security in neighborhoods that lost their "neighborhood public key" (say, because the adversary broke into too many nodes).

The partition impacts both security and performance of the protocol, and the designer should pick a partition that offers a good tradeoff. For many practical scenarios, a two-level solution, in which an n-node network is partitioned into $O(\sqrt{n})$ clusters of $O(\sqrt{n})$ nodes each, would give good tradeoff. However, if the original scheme can tolerate adversaries who break up to $n/2$ nodes, the resulting scheme can only tolerate adversaries who break up to $n/4$ nodes (since the adversary needs to compromise more than $\sqrt{n}/2$ neighborhoods, and compromising a neighborhood requires the compromise of at least $\sqrt{n}/2$ nodes).

There may exist other refinements to the scheme that handle scalability even better than partitions, at least in asymptotic terms. This is an interesting research problem.

Relaxations for small t. Note that Step 1 in protocol DISPERSE and Step 3 in protocol PARTIAL-AGREEMENT can be modified as follows, without affecting the agreement properties achieved: Instead of sending a message to all other parties, the sending node sends the message only to $2t + 1$ of the parties. Consequently, the complexity of these protocols (and of the entire proactive authenticator) is reduced from $O(n^2)$ to $O(nt)$ messages per node. This distinction may become significant when t is known to be small relative to n.

Acknowledgments

We wish to thank Mihir Bellare, Juan Garay, Rosario Gennaro, Oded Goldreich, Hugo Krawczyk, Tal Rabin, and Moti Yung for very helpful discussions and comments. We also thank the anonymous referees for their comments.

References

[1] D. Beaver. Foundation of secure interactive computing. In *Advances in Cryptology – CRYPTO* '91, volume 576 of Lecture Notes in Computer Science, pages 377–391. Springer-Verlag, Berlin, 1991.

[2] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the* 30*th Annual ACM Symposium on the Theory of Computing*, pages 419–428, 1998.

[3] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *Journal of the ACM*, 39(1):214–233, 1992. Extended abstract appeared in *Proc. STOC* '88.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO* '93, volume 773 of Lecture Notes in Computer Science, pages 232–249. Springer-Verlag, Berlin, 1993.

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

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[7] R. Bird, I. Gopal, A. Herzberg, P. A. 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, 1993.

[8] S. Blake-Wilson, D. Johnson, and A. Menezes. Key exchange protocols and their security analysis. In *Proceedings of the Sixth IMA International Conference on Cryptography and Coding*, 1997.

[9] S. Blake-Wilson and A. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proceedings of the 1997 Security Protocols Workshop*, 1997.

[10] R. Canetti. Security and composition of multiparty cryptographic protocols, *Journal of Cryptology*, this issue.

[11] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 639–648, 1996. Longer version available as MIT-LCS-TR 682, 1996.

[12] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: long-term protection against break-ins. *CryptoBytes*, 3(1):1–8, 1997.

[13] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In Yvo G. Desmedt, editor, *Advances in Cryptology – CRYPTO* '94, volume 839 of Lecture Notes in Computer Science, pages 425–438. Springer-Verlag, Berlin, 1994.

[14] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology – CRYPTO* '89, volume 435 of Lecture Notes in Computer Science, pages 307–315. Springer-Verlag, Berlin, 1989.

[15] W. Diffie, P. C. Van-Oorschot, and M. J. Weiner. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[16] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

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

[18] Y. Frankel, P. S. Gemmell, P. D. MacKenzie, and M. Yung. Optimal-resilience proactive public-key cryptosystems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 384–393. IEEE, New York, 1997.

[19] P. S. Gemmell. An introduction to threshold cryptography. *CryptoBytes*, 2(3):7–12, 1997.

[20] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *Advances in Cryptology - EUROCRYPT* '96, volume 1070 of Lecture Notes in Computer Science, pages 354–371. Springer-Verlag, Berlin,1996.

[21] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.

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

[23] A. Herzberg, M. Jakobson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proceedings of 4th ACM Conference on Computer and Communications Security*, 1997.

[24] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology - CRYPTO* '95, volume 963 of Lecture Notes in Computer Science, pages 339–352. Springer-Verlag, Berlin, 1995.

[25] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for the Internet. In *Proceedings of the IEEE Symposium on Network and Distributed System Security*, pages 114–127, 1996.

[26] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problems. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[27] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA, 1996.

[28] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology – CRYPTO* '91, volume 576 of Lecture Notes in Computer Science, pages 377–391. Springer-Verlag, Berlin, 1991.

[29] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the* 21*st Annual ACM Symposium on Theory of Computing*, pages 33–43, 1989.

[30] 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*, pages 51–59, 1991.

[31] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreements in the precence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[32] T. Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO* '98, volume 1462 of Lecture Notes in Computer Science, pages 89–104. Springer-Verlag, Berlin, 1998

[33] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the* 21*st Annual ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[34] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the* 22*nd Annual ACM Symposium on Theory of Computing*, pages 387–394, 1990.

[35] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *Proceedings of the* 23*rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE, New York, 1982.