Maintaining Authenticated Communication in the Presence of Break-ins*

Ran Canetti†
IBM T.J. Watson Research Center

Shai Halevi‡
IBM T.J. Watson Research Center

Amir Herzberg§
IBM Haifa Research Lab - Tel Aviv Annex

April 22, 1998

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, we want parties whose security is thus compromised to *regain* their ability to communicate in an authenticated way aided by other parties. In this work we present a mathematical model for this highly adversarial setting, exhibiting salient properties and parameters, and then describe a practically-appealing protocol for the task of maintaining authenticated communication in this model.

A key element in our solution is devising *proactive distributed signature (PDS) schemes* in our model. Although PDS schemes are known in the literature, they are all designed for a model where authenticated communication and broadcast primitives are 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 the above schemes, we also present a new definition of PDS schemes (and of distributed signature schemes in general). This definition may be of independent interest.

1 Introduction

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

Authenticated communication can be achieved using a variety of standard cryptographic techniques, even when the communication links are fully controlled by an adversary, provided that

*An extended abstract of this work appeared in the proceedings of Principles of Distributed Computing, 1997.

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

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

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

some cryptographic keys are distributed ahead of time (see, for instance, [5, 3, 24, 9, 8, 4]). 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 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. This approach, introduced in [28, 11], is aimed at maintaining security of cryptosystems in the presence of repeated break-ins. As in the "distributed cryptography" approach (see [18]), 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 securely carry out the task at hand (e.g., generating signatures) 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 [13] 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 which converts any Proactive Distributed Signature (PDS) scheme which works over reliable communication links into one that can work over faulty 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 the unreliable-links model. We present a formal definition of secure PDS schemes, and show how these can be constructed in our model. We note that our definition may be of independent interest. In particular, it may be used to define general distributed signature schemes (not just proactive ones), hence providing an alternative approach to the one in [19]. Finally we describe a method for transforming any protocol which works over reliable communication links into one that can work over faulty links, and analyze its properties. This method uses in an essential way the PDS schemes constructed before. We stress that a solution to this problem is essential for realizing any of the known proactive protocols (e.g., [28, 11, 23, 22, 16, 17, 30]) in a realistic system.

1.1 The model of computation

We consider a synchronous network of nodes that communicate via "totally unreliable" communication links and are also prone to break-ins. The faulty links are modeled by allowing an adversary full control over them. That is, the adversary can read, modify, delete and duplicate messages sent over the links, or even inject its own messages. Break ins are modeled by allowing the adversary to occasionally compromise nodes, thus obtaining all their secret data; even further, this secret data may be modified by the adversary. In this highly adversarial model, we would like to address the problem of recovering from break-ins and regaining authenticated communication. To make a solution possible we assume that the adversary is limited in the following three ways.

First, we must make sure that the adversary cannot modify the code of the recovery protocol itself (otherwise there is no hope for recovery). Therefore we assume that the **program** of each node is written in a Read-Only-Memory (ROM) and cannot be modified by anyone. (See discussion on the ROM assumption at the end of the Introduction.)

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. We thus adopt the proactive approach described above, introducing periodical, short **refreshment phases** during which the nodes jointly try to refresh their keys and regain security, and assuming some bound on the number of nodes which are broken into between any two successive refreshment phases. Below we refer to the time between two successive refreshment phases as a **time unit**. We consider adversaries that only breaks into a minority of the nodes in any time unit.

Third, note that a solution may still be impossible even if only a few nodes are broken at any given time, since the adversary can prevent a node from recovering by simply failing to deliver messages between this node and the rest of the network. To overcome this, we further assume a limit on the number and connectivity of links on which the adversary 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 towards dealing with this seemingly hopeless situation is as follows. First, we present a protocol which guarantees that the adversary must indeed 'cut off' N from a large fraction of the network in order to impersonate it, and hence a reasonably restricted adversary cannot impersonate too many nodes. Second, we make sure that a node will be *aware* that it is being impersonated by allowing it to keep a small piece of data in ROM, and using this piece of unmodifiable data as in order to 'bootstrap' the verification process. This piece of data (which is essentially a verification key of a signature scheme) is chosen at protocol startup, and remains unchanged and public throughout.

1.2 Defining our Goals

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

- *Authenticity.* This requirement ensures that a node will not accept as authentic an incoming message which was modified (or injected) by the adversary.
- *Delivery.* This requirement ensures that messages which are sent over the network arrive at their destination.

In most settings, these requirements can be dealt with separately. Indeed most of the authentication mechanisms in the literature only deal with ensuring authenticity (e.g., [2, 3, 4]), and explicitly ignore the delivery issue. The proactive setting is different, however, since recovering from a break-in requires that recovering nodes be able to communicate with other nodes in the network. 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 the sequel we refer to links which enjoy both authenticity and delivery as reliable links.

Defining 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 any protocol which is designed to work over reliable links into one that can work over faulty links.

In defining this notion we follow the general paradigm used for defining secure multiparty protocols [27, 1, 10]. That is, 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 we want to emulate: here the adversary has similar capabilities, with the exception that it must deliver messages faithfully on the links. Call this the authenticated-links (AL) model. The notion of "emulation" in also formulated in a standard way: 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 in 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 C as a compiler with the property that for any protocol π, the protocol $\pi' = C(\pi)$ emulates π in the UL model.

Awareness. Although our notion of emulation guarantees that π' and π have similar functionality from a 'global' point of view, it still allows the adversary to occasionally impersonate a node N. (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.

The power of the adversary. The above properties of a proactive authenticator only hold as long as the adversary is "appropriately limited". We limit the adversary both in the number of broken nodes and in the number and connectivity of "broken links". Capturing the power of an adversary (especially in the UL model) requires come care. Specifically, at any given time during the protocol we distinguish between three categories of nodes: A node can be either broken, 'disconnected' (when many of its links are tampered with), or else it is 'operational'. The adversary's power is thus captured using tree parameters: the number of nodes broken into between any two refreshment phases, the number (and connectivity) of links that have to be tampered with in order to 'disconnect' a node, and the number of disconnected nodes. See details within.

Related definitions. The authentication problem has been considered in a large number of works (e.g., [2, 6, 3, 5, 24, 9, 8, 4]). The approach taken in this paper (i.e., the notion of authenticators) was also adopted in [4]. Yet, the model in [4] 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 not to 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 inherent (to allow for joint refreshment phases). Finally, they emphasize the notion of independent *sessions* run by the same parties. Although we do not deal with session in this work, they can be incorporated here in a similar way as there.

1.3 Towards a solution

We take the following approach towards a solution: To obtain authenticity, at the beginning of each refreshment phase every node chooses at random a pair of signing and verification keys of some 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 sender's 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 over the (possibly faulty) links. Simply sending the verification key (perhaps signed using the old signing key) will not do: consider a node N just recovering from a break-in. N's old signing key is compromised. Thus, the adversary can forge N's signature and send a fake new verification key in the name of N, thus successfully impersonating N. Still, as we explained above, we want to design a mechanism which ensures a twofold task: On one hand it should be 'hard' to mount such an attack, and on the other hand such a node N must be aware of the fact that it is being impersonated.

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

As mentioned in Section 1.1, to counter such an attack we "bootstrap trust" during refreshment phases, *when a node cannot even trust its own memory*, by letting each node keep a small, unchanging public key in an unmodifiable ROM together with the code of the protocol.

1.4 Our Solution

The main technical tool in our solution is a proactive distributed signature scheme (PDS). Distributed signature schemes can be very roughly described as follows. 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 the nodes to jointly sign of a given message, while preventing the adversary from forging signatures, even after breaking into a few of these 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 signature key are changed periodically, the public verification key *remains unchanged throughout.*

All the PDS schemes known in the literature (e.g., [22, 19, 16, 17, 30]) 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 to work in our model (see Section 1.5). Once we have a PDS scheme in our model, our proactive authenticator proceeds as follows. First, we let each node keep a copy of the unchanging, public verification key of the 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 set of 'personal' signature and verification keys of some 'standard' (non-distributed) signature scheme.
- Obtain a certificate, generated jointly by all nodes using the PDS scheme, for the new verification key. (The certificate may read: *"it is certified that the personal verification key of N_i for time unit ℓ is v".*)

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

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

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

We show that this simple-to-implement construction meets our definition as long as the adversary does not break-into or disconnect more than half of the nodes during any single time unit.

1.5 Proactive signatures over faulty links.

To obtain a PDS scheme in our model of unauthenticated links, we describe 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. We note that although most schemes in the literature assume a broadcast channel (on top of reliable links), this assumption can be removed using standard agreement protocols [29, 25, 26, 14, 15], resulting in schemes which work over reliable point-to-point links. We therefore pick such schemes as a starting point for our solution.

Defining proactive signatures. To describe 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 satisfactorily definition, so we start by presenting one in this work. Our definition is quite general, and applies also to distributed (non-proactive) signature schemes. Here it can be used as an alternative to the [19] definition. It can also be regarded as a generalization of the [20] notion of (centralized) signature schemes existentially secure against chosen message attacks.

This definition too follows the usual paradigm for secure multi-party protocols. That is, we first formalize an ideal model for signature schemes. In this ideal model there are no signing and verification 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, where the notion of emulation is the usual one.

Realizing proactive signatures over faulty links. As we explained above, we assume as a starting point a PDS scheme that works in the AL model (i.e., with reliable and authenticated point-to-point links). Using simple tools we show how such a PDS scheme can be transformed into a PDS scheme in the UL model.

For this transformation, we need reliability of the links. 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 nodes local keys at the beginning of each refreshment phase.

We remark that for the purpose of obtaining a secure PDS scheme it is possible to use a simpler transformation than the one we present. Our transformation does more work in order to get extra properties needed to guarantee *awareness* of our proactive authenticator.

1.6 Discussion

The ROM assumption. Recall that for our solution we assume 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.

One particularly appealing method of implementing the ROM 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 enabling our solution.

In the absence of ROM in the nodes, one can try and use the network itself as a source of 'reliable memory', as described in [28]: 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.

The importance of awareness. The awareness condition is a somewhat non-standard aspect of our treatment, as it is a security requirement on the behavior of "non-functional" nodes. However, we view this condition as a crucial one and invest much effort in achieving it.

On one level, the awareness condition represents the fact that in our model it is impossible to completely prevent impersonation (e.g. in the case that a node is 'cut off' the network), and so we would like to ensure some measure of resistance even in these cases. We remark that in many settings, detecting an attack is "almost as good as preventing it". In the practice of computer security, it is well accepted that once an attack is detected, the system managers and security officers are well equipped to regain control and expose the attackers. Therefore the threat of an alert amy be sufficient to deter "hackers" from attacking the system.

On another level, the awareness condition is important in complementing our model assumptions: Recall that 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 tamper with too many links at the same time. In particular, in the sequel we show that it is important that the adversary does not inject messages on too many links at the beginning of a refreshment phase. In "real life", however, injecting messages on links is often much easier for an attacker than modifying existing messages, and this is often easier than breaking into nodes. Viewed in this light, the awareness condition ensures that the adversary gets "worse results" by mounting these easy attacks: although it can use them to impersonate a node, this node will be aware of it.

Can we tolerate stronger adversaries? A common problem in many distributed cryptographic protocols is that they must assume some limitations on the ability of the adversary to corrupt nodes in the network. Indeed, in most protocol, if the adversary can corrupt players beyond some pre-set bound, then "all bets are off", and the protocol cannot guarantee any security. This problem is even more acute in our solution, since we must also assume limitations on the ability of the adversary to tamper with communication links. As we said above, in many realistic scenarios is is easier to disrupt communication links than it is to break into nodes in the network. It is therefore 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 holds 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 can only partially guarantee this 'global awareness' property. Specifically, we can still guarantee awareness when the adversary is capable of injecting too many bogus messages on the links, but otherwise is not stronger than in our assumptions. As of yet, we do not know how to guarantee awareness in case the adversary can also modify or delete messages on too many links. This drawback could be overcome if we had a PDS scheme which is secure in a model where the links are authenticated but not necessarily reliable (i.e., where the adversary cannot modify, but can discard arbitrarily many messages on the links[1]). Yet, we do not know whether such PDS schemes exist; this is an interesting open problem.

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. Trusting other nodes to locally erase data (and, consequently, the proactive apprroach in general) may be problematic in a completely distributed setting where other nodes are not fully trusted. (See [12] for further discussion of this issue.)

Yet, it seems that in the setting of 'threshold cryptography', trusting other nodes in this manner

[1] 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.

may be reasonable. The reason is that typically, the nodes participating in such protocols are servers which may be under the same administrative domain (and are running the same code of the protocol), so it is as reasonable to trust remote nodes to locally erase data as it is to trust the local node to do it.

Scalability issues. To use our protocol in a very large network (e.g., the Internet), it is possible to partition the network to local neighborhoods, and then to perform the protocol only in local neighborhoods, with each neighborhood run its own PDS scheme. A node will obtain a certificate from one or more of the neighborhoods it belongs to. (The unchanging verification keys of all the neighborhoods can be signed — at system start-up — by a global certification authority.) The partition will impact 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, which involved partitioning the n-node network 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 upto $n/2$ nodes, the resulting scheme can only tolerate adversaries who break upto $n/4$ nodes. There may exist other refinements to the scheme that will handle scalability even better than partitions, at least in asymptotic terms. This is an interesting research problem.

1.7 Road map

The rest of this paper is organized as follows. In Section 2 we formally define the two main computational models discussed above, namely the authenticated links (AL) model and the unauthenticated links (UL) models. We also define the central primitive of this paper: proactive authenticators. In Section 3 we define proactive distributed signature (PDS) schemes, both in the AL model and in the UL models, and in Section 4 we show how to construct a PDS scheme in the UL model, given any PDS scheme in the AL model. These two sections are of independent interest, and can be read separately of the notion of authenticators (i.e., skipping Sections 2.3 and 5). Finally, in Section 5 we construct and prove the security of the proactive authenticator described above.

The bulk of the work is invested in defining, constructing, and proving the security of a PDS scheme in the UL model. Proving the security of our authenticator (given such a PDS scheme) is done very similarly to the proof of security of the PDS scheme itself.

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 identities of broken nodes are chosen 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).

2.1 The Authenticated Links (AL) Model

We consider a *synchronous* network of n nodes, $N_1, ..., N_n$, 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 $x_i = x_{i,1}, x_{i,2}, \dots$ where the piece $x_{i,l}$ is given to node N_i at the onset of communication round l. Also, to model the ability of nodes to make fresh random choices, we think of r_i as consisting of pieces $r_i = r_{i,1}, r_{i,2}, \dots$ where the piece $r_{i,l}$ is chosen by node N_i only at the onset of communication round l. In addition to x_i and r_i, each node is given n, the number of nodes, and a security parameter k (both n and k are encoded in unary, to let the nodes 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 broken nodes at wish. 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 unauthenticated-links 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 also allow a protocol to have 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 setup phase can be replaced by an execution of a centralized setup 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 in some canonical way 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.[2] See Figure 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. (The durations of time units and refreshment phases can be thought of as parameters of the model; a given protocol fits the model with specific values for these parameters. For simplicity we leave these parameters implicit in the presentation.)

Definition 1 **(t-limited adversary)** *An AL-adversary $\mathcal{A}$ is called t-limited with respect to protocol π if on any inputs and randomness 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.

[2]The reason for the name is that the nodes will typically "refresh their cryptographic keys" during this overlap.

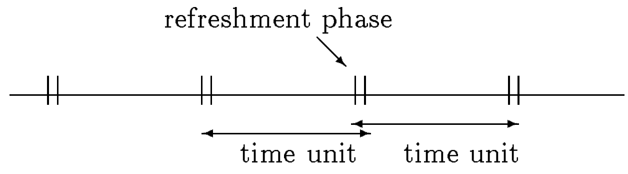

Figure 1: Time units and refreshment phases.

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 $\pi, \mathcal{A}$ 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. 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 the sequel 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 regards 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, each non-broken node outputs whatever it 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.[3] The global output of the computation up to round l is the concatenation of the local outputs from all rounds of the all nodes and adversary. When dealing with protocols for specific tasks it may be convenient to let the global output contain additional information which is relevant to the functionality of protocols. 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 0th 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.

Possible generalizations. The syntax above can be generalized in a few ways. For example, one can address on-going executions that do not terminate by defining prefixes of executions and global outputs up to a given communication round. Also, one can let the adversary and nodes have some arbitrary auxiliary information (on top of their inputs and randomness), capturing information gathered from other protocols run by the nodes. This may be important for secure composition of

[3] 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.

protocols; see discussion in [10]. Yet another generalization deals with interactive inputs, by letting the inputs of the nodes in round ℓ depend on the execution of the protocol upto this round (this dependence may or may not be under the control of the adversary, depending on the context of the protocol).

2.2 The Unauthenticated-Links (UL) Model

The unauthenticated-links (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. Second, we allow the nodes to set a special small ROM at the end of the set-up phase. As described 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 also here the adversary remains inactive during the set-up phase.) Transcripts and adversary views are defined as in the AL model. We let $\text{UL-TRANS}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})$ and $\text{UL-TRANS}_{\pi,\mathcal{U}}(k,\vec{x})$ to have analogous meaning, in the UL model, to $\text{AL-TRANS}_{\pi,\mathcal{A}}(k,\vec{x},\vec{r})$ and $\text{AL-TRANS}_{\pi,\mathcal{U}}(k,\vec{x})$.

Outputs are defined a bit differently, as described at the end of this section.

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 2 (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*[4] *if during this time:*

(a) *Neither N_i nor N_j are 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, and N_i, N_j do not receive any other message on this link.*

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

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

[4]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.

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

Definition 3 (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 N 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, and*

   (b) *During this round it has reliable links with at least $n-s+1$ nodes which 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, and*

   (b) *There is a set S, consisting of at least $n-s+1$ nodes which are s-operational throughout this refreshment phase, such that N has reliable links with all the nodes in S throughout this refreshment phase.*

Definition 4 (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, and 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 5 (**(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.*

The output of an execution. The output of an execution is defined as in the AL model, except that here, the line `Node $N_i$ is compromised' is added to $N_i$'s output not only when it is broken, but also when it becomes $s$-disconnected, and the line `Node N_i is recovered' is added to N_i's output only when it becomes s-operational again.[5] We let $\text{UL-OUT}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})$ and $\text{UL-OUT}_{\pi,\mathcal{U}}(k,\vec{x})$ 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.

2.3 Proactive authenticators

In this section we define the requirements from a "compiler" that transforms protocols which are designed to run over a network with authenticated links (and repeated break-ins) into protocols which can run over a network with *un*authenticated links (and repeated break-ins). We call such compilers proactive authenticators.

Syntactically, an n-node authenticator takes for input a description of a protocol (for n nodes) in the AL model and outputs a description of another protocol (for n 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 multi-party protocols [27, 1, 10]. We complement this requirement with another one, called awareness. Awareness roughly means that a node will notice whenever it is being 'impersonated' by the adversary.

Definition 6 (**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 for all input vectors $\vec{x}$*

$$\text{AL-OUT}_{\pi,\mathcal{A}}(k,\vec{x}) \stackrel{c}{\approx} \text{UL-OUT}_{\pi',\mathcal{U}}(k,\vec{x}) \tag{1}$$

Where $\stackrel{c}{\approx}$ denotes 'computationally indistinguishable'.[6]

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 7 *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.*

[5] In this model the definition of the protocol's output depends on the parameter s. We can therefore have different definitions for the output of the protocol, depending on the properties which we want to prove about this protocol.

[6] Two distribution ensembles $\{A_k\}_{k\in\mathrm{N}}$ and $\{B_k\}_{k\in\mathrm{N}}$ are computationally indistinguishable if for every polynomial-time algorithm D, for all polynomial p and all sufficiently large k, $|\mathrm{Prob}(D(A_k)=1)-\mathrm{Prob}(D(B_k)=1)| < \frac{1}{p(|k|)}$.

Awareness. While simulatability guarantees that the the "global" behavior of the system remains unchanged in the UL model, it does allow a limited number of nodes to be disconnected from the rest of the network, and consequently be impersonated by the adversary. As discussed in the Introduction, 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 guarantees may still be valuable; in particular, if the nodes operate in an environment where a higher level protocol or a human operator which can use out-of-band communication to restore security, then detecting an attack may be nearly as good as preventing it.

At first glance it may seem that talking about anything other than the global protocol behavior is irrelevant in the context of distributed protocols, but there are many setting in which this is not true. As a case example, consider a "proactivized operating system", in which several nodes execute some protocols to secure the usual operating-system services such as access to file system, printers, networks, etc. In addition to the operating system itself, there are typically many other applications running on these nodes, some of which are distributed and some are not. In this case, we may also care about the behavior of the non-distributed applications, hence we would like to consider the behavior of individual nodes in the network.

We start by defining 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 8 (internal and external views, impersonation) *Let Λ be an n-node layered authenticator and let π be an n-node protocol, and consider an execution of $\Lambda\pi$ in the presence of an 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 a N_i during u is the sequence of all the messages which were sent and received by the top layer (i.e., by protocol π) during u.*

The external view *of N in time unit u consists of all the messages which appear, in the internal views of other non-broken 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.*

Definition 9 *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 $\mathcal{U}$, any node which impersonated outputs a special alert signal in the time unit in which it is impersonated, except with a negligible probability (in the security parameter k).*

3 Defining Proactive Distributed Signature Schemes (PDS)

Our main technical tool in this paper 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 [20] notion of security against existential forgery in the presence of chosen message attack: 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 of this message. We instantiate this general paradigm into definitions for secure PDS schemes in the AL and UL models. The general paradigm presented here, as well as the particular instantiations, may well be of independent interest.

Our definition follows 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 (standard) sense used in 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 [20] definition. At the end of this section (Remark (5)) we briefly discuss an alternative approach towards defining secure distributed signature schemes, taken in [19].

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

Let us 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 will be able to distinguish between messages which were properly signed and messages which were not. More precisely, we need the following functionality:

Threshold. A message is signed only if a large enough subset of the signing parties 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 signing parties

We view the signatures attached to messages and the keys used to verify these signatures merely as tools for obtaining this functionality. Indeed, 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 signing parties. Verification whether a message is signed is done simply by checking whether or not this message appears in the database.

More precisely, in the ideal process there are n signers $N_1 \dots N_n$ and a signature verifier V.[7] The parties communicate with a trusted party T and a probabilistic-polynomial-time ideal-model forger $\mathcal{IF}$. There is no communication among the parties. The process is parameterized by t, the threshold

[7] The signature verifier V is used to ensure that signed messages can be verified without interacting with the signers.

for signing messages, and by a security parameter k.[8] In addition, the forger and each signer N_i has some external input; this input is not used directly by the signing parties; its 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. (It is supposed 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 hands a request "sign (m, u)" to the trusted party T, and appends a note \``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 signers, and these requests agree on m and u, it adds (m, u) to the set M of signed messages. In addition each signer that asked to sign (m, u) receives a note \``(m, u) is signed`' from T, and appends this note 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 note \``N_i is compromised`' is appended to N_i's output, and the forger learns the signer's 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 a note saying \``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 note to its output. Otherwise $V$ responds with \``m is not verified`', but *does not add anything to its output.* See Remark (4) 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.

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

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

denotes the outputs from an execution of the system above and the adversary $\mathcal{IF}$ on randomness $\vec{r}$, input $\vec{x} = x_{\text{F}}, x_1, ..., x_n$ 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 $\text{ID-SIG}_{t,\mathcal{IF}}(k,\vec{x},\vec{r})_V$.) We denote by $\text{ID-SIG}_{t,\mathcal{IF}}(k,\vec{x})$ the distribution of $\text{ID-SIG}_{t,\mathcal{IF}}(k,\vec{x},\vec{r})$ when $\vec{r}$ is uniformly distributed.

[8]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.

3.2 The structure and operation of PDS schemes

We now turn to describe the structure and operation of PDS schemes, either in the AL or the UL model. We also introduce syntax necessary for the definition of security.

Structure. 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.

- The key-generation protocol, Gen, is run in the setup phase of the scheme. Each node is given the security parameter k (encoded in unary), and outputs a public key pk and a share sk_i of a secret key.

  Since this phase is executed in an "adversary free" environment, we can assume that at the end of this protocol all the nodes have the same public key. Below we also assume without loss of generality, that k is implicit in the keys generated by $\mathsf{Gen}(k)$.

- In the signing protocol Sign each signing node N_i is given the public verification key pk and its share sk_i of the private signing key (sk_i is known only to N_i.) When invoked, N_i interacts with the other nodes and eventually outputs a value σ. (Hopefully, σ is a signature on m, verifiable by the public key pk.)

- The verification algorithm Ver is given the public verification key pk, a message m and an alleged signature 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 a share sk_i of the signing key, the public key pk, and potentially some other state information. The nodes interact and eventually each N_i outputs sk'_i, a new share of the signing key.

Operation. The operation of 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 signing nodes $N_1, ..., N_n$, the adversary $\mathcal{F}$ and an unbreakable signature verifier V.[9] Also here, the nodes have (auxiliary) input which they ignore.

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

Signatures. The adversary (in either model) can send a special message 'sign m' to any of the 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 (m, u). 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 (m, u) (with respect to pk) then the line `` `$(m, u)$ is signed´ `` is added to its output.

Verification. The adversary $\mathcal{F}$ can invoke the signature verifier V with a pair (μ, σ). V runs algorithm Ver on $(\mu, \sigma, \mathsf{pk})$; if the verification succeeds then V outputs `` `$\mu$ is verified´ ``. If the verification fails then it outputs nothing (see Remark (4) at the end of this section). We stress that V cannot be broken into.

[9] 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.

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 the 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 $\mathsf{sk}_{i,u-1}$ 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 the point where the adversary halts. 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 of the verifier V.

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

$$\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r}) = \\ \langle \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_A, \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_1 \ldots \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_n, \text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}, \vec{r})_V \rangle$$

denotes the outputs from an execution of the scheme $\mathcal{S}$ with the AL-model adversary $\mathcal{AF}$ on randomness $\vec{r}$, inputs $\vec{x} = x_{\text{F}}, x_1, \ldots, x_n$ 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.

The notations $\text{UL-SIG}_{S,\mathcal{UF}}(k, \vec{x}, \vec{r})$ and $\text{UL-SIG}_{S,\mathcal{UF}}(k, \vec{x})$ are defined similarly with respect to a UL-forger $\mathcal{UF}$ and scheme S.

3.3 Security of a PDS scheme

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

Definition 10 *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 forger $\mathcal{IF}$ such that for all inputs $\vec{x}$,*

$$\text{AL-SIG}_{S,\mathcal{AF}}(k, \vec{x}) \stackrel{c}{\approx} \text{ID-SIG}_{t+1,\mathcal{IF}}(k, \vec{x}) \tag{2}$$

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 for all inputs $\vec{x}$,*

$$\text{UL-SIG}_{S,\mathcal{UF}}(k, \vec{x}) \stackrel{c}{\approx} \text{ID-SIG}_{t+1,\mathcal{IF}}(k, \vec{x}) \tag{3}$$

Notice that in the definitions above we set the threshold of the PDS schemes to $t+1$ (where t is the upper bound on the number of nodes the adversary can compromise in every time unit). In principle it may make sense to set the threshold to any value between $t+1$ and $n-t$, but this extra generality is not really needed for our purpose.

3.4 Discussion

(1) As in the case of Definition 6, requirements (2) and (3) incorporate many conditions. Here they imply, in particular, that in the real-life computation the verifier does not accept a message as signed, unless at least $t+1$ signers were invoked with a 'sign m' request. Also, they imply that whenever at least $t+1$ "good nodes" invoke the signature protocol on the input m, then all these good nodes obtain a valid signature on m.

In fact, 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, but only about its (in)ability to forge signatures. We only kept it here to be consistent with our general framework, and because our constructions can withstand this stronger requirement.

(2) Note that there is no need to explicitly limit the number of break-ins made in the ideal model, since in order to successfully imitate a real-life interaction the ideal forger must in particular break into the same nodes that are broken in a real-life interaction.

(3) Our definition requires that, in order to generate a signature on a message m, the adversary will request at least $t+1$ signers to sign m *within a single time unit.* This particular 'granularity' is convenient for our application. Yet, in principle the granularity can be changed as appropriate.

(4) We motivate our choice to have V output nothing in case that 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 hands 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 signatures, 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 that 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.

(5) A different approach to defining distributed signature schemes is taken in [19]. There, one starts with some particular centralized signature scheme (in the case of [19] this is the DSS scheme), and regards its key-generation and signing algorithms as probabilistic functions. Next, one makes sure that the distributed key-generation and signing protocols of the threshold scheme securely evaluate the corresponding functions, according to some standard definition of secure multi-party 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. This 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.

4 Constructing PDS schemes in the UL model

We use the following basic result as our starting point:

Theorem 11 *If trapdoor permutations exist, then for any $n > 2t+1$ there exist n-node t-secure PDS schemes in the AL model.*

The theorem can be proven, for instance, using the general techniques of [32, 7, 31, 28, 12, 14]. ([32] construct a signature scheme from any one-way function. [7, 31] show how to make this scheme into a threshold one, assuming secure channels and a broadcast channel. [28] show how to turn this threshold scheme into a proactive one. [12] show how to transform the scheme into one in the AL model by using encryption on the links. [14] show how to implement a broadcast channel using centralized signature schemes.) Furthermore, [22, 19, 16, 17, 30] show how to construct 'practical' secure PDS schemes in the AL model from many centralized signature schemes such as ElGamal, DSS and RSA. (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.) 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. In addition, the constructed scheme will have properties that will allow the authenticator built on top of it in Section 5 to maintain awareness.

Ingredients. The construction uses the following algorithms and protocols:

- An n node PDS schemes in the AL model, denoted by $\mathsf{ALS} = \langle A\mathsf{Gen}, A\mathsf{Sign}, A\mathsf{Ver}, A\mathsf{Rfr}\rangle$, which is t-secure in the sense of Definition 10.
- A centralized signature scheme, denoted by $\mathsf{CS} = \langle C\mathsf{Gen}, C\mathsf{Sign}, C\mathsf{Ver}\rangle$ which is existentially unforgeable under an adaptive chosen message attack [20].

We use the schemes ALS and CS to construct an n node PDS scheme in the UL model, which we denote $\mathsf{ULS} = \langle U\mathsf{Gen}, U\mathsf{Sign}, U\mathsf{Ver}, U\mathsf{Rfr}\rangle$, and then use the security of $\mathsf{ALS}, \mathsf{CS}$ to prove that ULS is (t, t)-secure.

Informally, 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 some additional steps during each refreshment phase.

The rest of this section is organized as follow: In Section 4.1 we describe some tools used in the construction. These are simple protocols used to 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. In this section we describe the protocols, give some intuition for their use, and prove a few basic properties about them. Later in this section we describe how these protocols fit in our construction and its proof of security.

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 which guarantees that the adversary must disrupt many links to prevent messages from being delivered. The protocol, which we call DISPERSE, is just a 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" all other nodes.*
2. *Upon receiving a message "forward m to N_j", allegedly from N_i, each node N_k sends the message "forwarding m from N_i" to the N_j.*
3. *Node N_j marks all the strings m for which it received a message "forwarding m from N_i to N_j" as being received from N_i.*

Figure 2: Code of the DISPERSE protocol.

"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 re-transmitted. The only guarantee it offers is that if N_i, N_j are two nodes so that there is a path in the network between N_i and N_j of length at most two consisting only of reliable links, then any message which is sent between these nodes using the DISPERSE protocol arrives at its destination. In conjunction with our definition of operational nodes (Definition 3) we get the following lemma

Lemma 12 *Let n be the number of nodes in the network and let $s \leq \lfloor \frac{n-1}{2} \rfloor$. If N_i, N_j are two s-operational nodes in some time interval, then during this time interval N_j receives every message which N_i sends to it using the* DISPERSE *protocol.*

Proof: According to Definition 3, if N_i, N_j are s-operational, then they both have reliable links to at least $n - s > \frac{n}{2}$ other s-operational (and hence, non-broken) nodes. Therefore there must exists at least one non-broken node which has 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 any of them (and without even modifying any message on the direct link between them). To obtain some authenticity guarantees, we combine the DISPERSE protocol with digital signatures. Digital signatures require each node N_i to have the following keys:

(a) A pair of signature and verification keys for the centralized signature scheme CS. Below we denote these keys by s_i^u and v_i^u, respectively, and refer to them as the *local keys* of N_i.

(b) A public verification key for the PDS scheme ALS, which we denote by v_{cert} and refer to as the the *global verification key.*[10]

(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.

[10]Recall that in our network model we assume that all the nodes share the same global verification key.

algorithm $\textsc{Certify}(m, i, j, u, w, s_i^u, v_i^u, \mathsf{cert}_i^u)$

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

algorithm $\textsc{Ver-cert}(j, i, u, w, \mathtt{msg}, v_{\mathsf{cert}})$

1. *Check format: parse* $\mathtt{msg} = \langle m, i', j', u', w', \sigma, v, \mathsf{cert}\rangle$. *If* $i' \neq i$ *(wrong source),* $j' \neq j$ *(wrong destination) or* $u' \neq u$ *or* $w' \neq w$ *(wrong time) then reject* $\mathtt{msg}$.
2. *Verify certificate: execute* $A\mathsf{Ver}$ *('the public key of* N_i *in time unit* u *is* v*',* $\mathsf{cert}, v_{\mathsf{cert}}$*). If verification fails, reject* $\mathtt{msg}$.
3. *Verify signature: execute* $C\mathsf{Ver}(\langle m, i, j, u\rangle, \sigma, v)$. *If verification fails, reject* $\mathtt{msg}$.
4. *If both verifications succeed, accept* $\mathtt{msg}$ *as coming from* N_i.

Figure 3: The Certify and Ver-cert algorithms

protocol $\textsc{Auth-send}\,(m, i, j, u, s_i^u, v_i^u, v_{\mathsf{cert}}, \mathsf{cert}_i^u)$

Sending a string m *from* N_i *to* N_j *in communication round* w *during time unit* u *using local keys* $s_i^{(u)}, v_i^{(u)}$*, global verification key* v_{cert} *and certificate* cert_i^u

1. *Node* N_i *sets* $\mathtt{msg} = \textsc{Cerfity}(m, i, j, u, w, s_i^u, v_i^u, \mathsf{cert}_i^u)$*, and invokes protocol* $\textsc{Disperse}(\mathtt{msg}, i, j)$.
2. *for every message* $\mathtt{msg}$ *which was received from* N_i*, node* N_j *invokes* $\textsc{Ver-cert}(j, i, u, w, \mathtt{msg}, v_{\mathsf{cert}})$ *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).*

Figure 4: Code of the Auth-send protocol.

protocol $\textsc{Partial-agreement}(i, x_i, u, w, s_i^u, v_i^u, v_{\mathsf{cert}}, \mathsf{cert}_i^u)$

Node N_i starts with input value x_i in communication round w of time-unit u and uses local keys s_i^u, v_i^u, global key v_{cert} and certificate cert_i^u.

1. *For every other node N_j, node N_i invokes* $\textsc{Auth-send}(x_i, i, j, u, v, s_i^u, v_i^u, \mathsf{cert}_i^u, v_{\mathsf{cert}})$.
2. *N_i collects all the input values which 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 which are not marked "cheaters", so that $|\mathsf{MAJ}_i| \geq \lceil \frac{n+1}{2} \rceil$ 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 $y_i = \phi$, and $\mathsf{MAJ}_i = \emptyset$.
3. *For any certified message* $\mathtt{msg}$ *which was accepted in Step 1 from a node in MAJ_i, node N_i invokes* $\textsc{Disperse}(\mathtt{msg}, i, j)$ *to distribute* $\mathtt{msg}$ *to all other nodes N_j.*
4. *N_i applies verification procedure* $\textsc{Ver-cert}(i, j, u, w, \mathtt{msg}, v_{\mathsf{cert}})$ *for each message* $\mathtt{msg}$ *which 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".*

   Denote by MAJ'_i the set of nodes which were in MAJ_i in Step 2, and were not marked "cheaters" in Step 4. If MAJ'_i still contains at least $\lceil \frac{n+1}{2} \rceil$ nodes, then y_i is the output value of N_i. Otherwise, N_i outputs ϕ.

Figure 5: Code for protocol $\textsc{Partial-agreement}$

These keys are used in the 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 $\langle m, i, j, u, v_i^u \rangle$, and attaches to it 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 v_{cert} to verify the signatures on the certificate and then verify the signature on the message. More precisely, we have algorithms $\textsc{Certify}$ and $\textsc{Ver-cert}$ which are described in Fig. 3, and protocol $\textsc{Auth-send}$ which is described in Fig. 4. In the sequel we sometimes use the term *properly certified message* to describe a message that passes the verification algorithm. (Here we assume that each node always has *some* values for its keys, regardless of whether these values are valid of not.)

Partial agreement. Our construction needs to have the nodes agree on the value to be signed. To that end we use the following standard $\textsc{Partial-agreement}$ protocol. Intuitively, the goal of this protocol is to ensure that there is a single value x such that every node either ends up with the value x, or end 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 13.

Lemma 13 *Assume that during some execution of protocol* $\textsc{Partial-agreement}$ *there exists a set S of non-broken nodes of size at least $\lceil \frac{n+1}{2} \rceil$ 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* $\textsc{Disperse}$ *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 the execution E with the same input value x, then they all output x as the output value.*

2. *If one of the nodes in S outputs a value $x \neq \phi$, then all the nodes in S output either x or ϕ.*

Proof: 1. Since each node $N_i \in S$ only certifies a single input value, and the messages which 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 $|\mathsf{MAJ}'_i| \geq |S| \geq \lceil \frac{n+1}{2} \rceil$ 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 $y_i \neq \phi$, and $N_j \in S$ outputs $y_j \neq y_i$ (and also $y_j \neq \phi$). Since both MAJ'_i and MAJ'_j contain at least $\lceil \frac{n+1}{2} \rceil$ 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 $n > 2t$. We construct an n-node PDS scheme $\mathsf{ULS} = (U\mathsf{Gen}, U\mathsf{Sign}, U\mathsf{Ver}, U\mathsf{Rfr})$ in the UL model, given a centralized signature scheme $\mathsf{CS} = (C\mathsf{Gen}, C\mathsf{Sign}, C\mathsf{Ver})$ and an n node PDS scheme in the AL model $\mathsf{ALS} = (A\mathsf{Gen}, A\mathsf{Sign}, A\mathsf{Ver}, A\mathsf{Rfr})$.

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. More precisely, the scheme ULS proceeds as follows.

4.2.1 Key generation, $U\mathsf{Gen}$

On security parameter k node N_i first executes $A\mathsf{Gen}$, the key generation protocol of ALS, to obtain the public key v_{cert} and the secret share sh_i^0. Next N_i executes the key generation algorithm of CS to obtain $\langle v_i^0, s_i^0 \rangle \leftarrow C\mathsf{Gen}(k)$. 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

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

Node N_i then records v_{cert} in its read-only memory, and stores $(\mathsf{sh}_i^0, v_i^0, s_i^0, \mathsf{cert}_i^0)$ in regular memory. 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.

Throughout the protocol, during time unit u each node N_i has local keys v_i^u, s_i^u, share sh_i^u and certificate cert_i^l 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 which 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 $s_i^u, v_i^u, \mathsf{cert}_i^u$, to send m.

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, and in the second part the nodes execute the refreshment protocol $A\mathsf{Rfr}$ of the PDS scheme ALS.

Part (I). This part should be thought of as taking place "at the end of the previous time unit", since during this part node 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 $(s_i^u, v_i^u) \leftarrow C\mathsf{Gen}(k)$.

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. A node N_j, which receives a key v_i allegedly from N_i, invokes a copy of PARTIAL-AGREEMENT with v as the input, and using the keys from the time unit $u-1$, to get

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

   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_i allegedly from N_i, then N_j runs PARTIAL-AGREEMENT on the first such value.

4. If $v_{i,j} \neq \phi$, 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 $v_{i,j}$". 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 receive from any node a valid signature with respect to v_{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 $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$. 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" 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 which is sent from N_i to N_j, the sender in $U\mathsf{Rfr}$ invokes AUTH-SEND, using its new keys $s_i^u, v_i^u, \mathsf{cert}_i^u$, 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 signature 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 $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$, or has failed to refresh its share during Part (II), then N_i outputs `alert'.[11]

Remarks. **1.** In the definition of reliable links (Definition 2) 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 for 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'.

2. We remark that Step 3 of the refreshment phase (i.e., protocol PARTIAL-AGREEMENT) is not necessary for proving that ULS is (t,t)-secure. Its only purpose is to allow the authenticator built on top of ULS to maintain awareness (see Section 5).

4.3 Proof of security

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 which is 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.*

Proof overview. Before presenting the formal proof, let us give a high-level overview of the main technical parts in it. According to Definition 10, to prove that ULS is (t,t)-secure in the UL model we need to show that every (t,t)-limited UL-forger which interacts with ULS can be simulated by an ideal forger with threshold $t+1$. Using our assumption that ALS is t-secure in the AL model, it is sufficient to show that every (t,t)-limited UL-forger which interacts with ULS can be simulated by a t-limited AL-forger which interacts with ALS, since we know that the later can be simulated by an ideal forger. In the proof below we therefore fix an arbitrary (t,t)-limited UL-forger $\mathcal{UF}$ and describe a t-limited AL-forger $\mathcal{AF}$ interacting with ALS that satisfies (for all inputs $\vec{x}$):

$$\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \stackrel{c}{\approx} \text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x}). \tag{4}$$

(In fact, we will show that the *statistical distance* between the two sides of (4) is negligible.) As usual, we define $\mathcal{AF}$ via a simulation of $\mathcal{UF}$. To prove Eq. 4 we define "good executions" as those where no messages are forged by the adversary $\mathcal{UF}$, and where all the operational parties have valid certificates for their local keys. Then we show that good executions have the same probability weight according to both $\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x})$ and $\text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x})$, and that executions which are not good have only a negligible probability weight. The first assertion follows almost immediately from the way we define the simulator $\mathcal{AF}$. The second part is proven via a sequence of claims,

[11]We do not use this alert signal anywhere in the security proof below; it is used later in Section 5.

roughly sketched below. Consider a partial execution E which is good upto the end of time unit $u-1$:

- From the properties of PARTIAL-AGREEMENT is follows that if N_i is operational in time unit u, then in the refreshment phase at the beginning of this time unit there are many operational nodes which participate in signing N_i's local public key.
- It follows from the security of the ALS scheme, 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.
- 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.
- 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. But 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.

We then conclude that if E is also a good execution upto the end of time unit u, except with negligible probability. Details follow. (We remark that the proof below does not proceed by induction, but instead uses $\mathcal{AF}$ 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.)

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 $\mathcal{AF}$ which simulates $\mathcal{UF}$. On a high level, $\mathcal{AF}$ executes the algorithm $\mathcal{UF}$ as a black-box, emulating the UL-model network that $\mathcal{UF}$ expects to interact with. To carry out this emulation, $\mathcal{AF}$ 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 to the UL-network that $\mathcal{AF}$ emulates for $\mathcal{UF}$ we refer to as the 'emulated network'. Similarly, we distinguish between 'real communication rounds' in the real network and 'emulated communication round' in the emulated network (typically we need several rounds of the emulated 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 emulated network are called 'emulated 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 (both encoded in unary), its input x_{F}, a public key v_{cert} (generated in $A\mathsf{Gen}(k)$), and access to the real network where n nodes are running the protocol ALS with the secret keys corresponding to v_{cert}.

$\mathcal{AF}$ then generates the following keys: For each emulated node N_i it runs a copy of the key-generation algorithm of CS to obtain $\langle v_i^0, s_i^0 \rangle \leftarrow C\mathsf{Gen}(k)$. Then, for every i it asks all the nodes

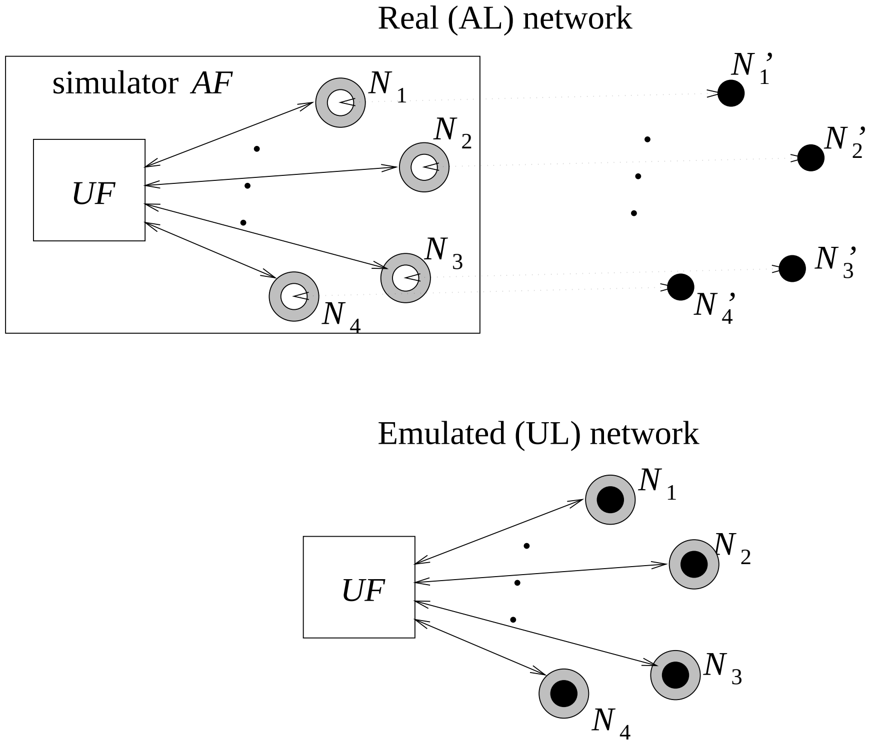

Figure 6: Operation of the simulator $\mathcal{AF}$. The solid areas represent the upper layer of ULS(ie, the scheme ALS). The grey areas represent the lower layer of ULS.

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 signature will be generated when the AL-model nodes execute the ASign protocol, since they all receive the same message to sign and none of them is broken. Finally, $\mathcal{AF}$ initializes $\mathcal{UF}$, giving it $k, n, x_{\mathtt{F}}$ and v_{cert}. Throughout the execution, during time unit u $\mathcal{AF}$ maintains for each emulated 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 emulated 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 of the emulated nodes 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 emulated nodes. For each emulated node N_i, $\mathcal{AF}$ keeps a status variable which can be either 'operational' – if N_i is t-operational in the current emulated 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 emulated 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.[12]

[12]This reflects the intuition that since N_i is disconnected, it may appear to be broken in the eyes of the other nodes

If $\mathcal{UF}$ asks to break an emulated 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 emulated N_i. When $\mathcal{UF}$ asks to change the contents of the memory of a broken emulated 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 emulated node N_i, then $\mathcal{AF}$ change the status of N_i from 'broken' to 'disconnected', *but does not leave the real node N'_i yet.* $\mathcal{AF}$ leave N'_i only when the emulated node N_i becomes 'operational'.

Real communication rounds. At the beginning of every real communication round, $\mathcal{AF}$ records all the messages which were sent by the non-broken nodes in the real network. For every message m that the non-broken real node N'_i sends to real node N'_j, $\mathcal{AF}$ emulates an execution of the protocol AUTH-SEND, among the emulated nodes, using the local keys v_i^u, s_i^u, cert_i^u that it maintains for the emulated N_i, and the global verification key v_{cert}.

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

If an emulated node N_j which is marked 'disconnected' accepts a message string m from another emulated 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 emulated node N_j accepts a message string m from another emulated node N_i which 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 emulated node N_j which is marked 'operational' accepts a message string m from another emulated node N_i which 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.

Emulated signatures. When emulated node N_i which is not 'broken' accepts 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.

Emulated refreshment phases. Below we require that the emulated refreshment phases begins before the refreshment phases in the real network[13], so that Part (II) of the refreshment protocol $U\mathsf{Rfr}$ is aligned with the refreshment protocol $A\mathsf{Rfr}$.

At the beginning of an emulated refreshment phase in time-unit u, $\mathcal{AF}$ emulates an execution of Part (I) of the refreshment phase $U\mathsf{Rfr}$. This is done by executing the following procedure for every emulated node N_i which is not marked 'broken':

1. Execute a copy of the key-generation algorithm of CS to obtain $\langle v_i^u, s_i^u \rangle \leftarrow C\mathsf{Gen}(k)$.
2. Send v_i^u to all the emulated nodes. This amounts to giving $\mathcal{UF}$ the message v_i^u on behalf of the emulated node N_i, and ask that this message be delivered to all the nodes.

even if it is not really broken.

[13]This means that the time units in the emulated network starts slightly before the time units in the real network.

3. For each emulated node N_j which 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 emulated 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 emulated invocation of PARTIAL-AGREEMENT there exist two emulated 'operational' nodes with two different local output values, both different from ϕ, then $\mathcal{AF}$ outputs "failure" and halts.

4. For every emulated node N_j which is not 'broken', if the local output of N_j from PARTIAL-AGREEMENT was $x_j \neq \phi$ 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 x_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 x_j then $\mathcal{AF}$ asks $\mathcal{UF}$ to deliver this value to emulated node N_i on behalf of emulated node N_j.

6. If $\mathcal{UF}$ delivers to emulated 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 $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$.

When this process if over, $\mathcal{AF}$ emulates 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 emulated 'operational' node N_i with $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$, 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 $\mathcal{AF}$ will be 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 which $\mathcal{AF}$ breaks equals the number of nodes which are not t-operational in the emulated 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.

Next, we 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 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 between an execution and its transcript.)

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

$$\text{AL-SIG}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \overset{s}{\approx} \text{UL-SIG}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x}) \tag{5}$$

where $\stackrel{s}{\approx}$ denotes 'having negligible statistical distance'. To prove this, we identify a set GOOD of "good executions", and demonstrate the following two facts.

Fact 1: Conditioned on the set GOOD, the above two distributions (on the global outputs of executions) are identical. Moreover, 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 which are not good is negligible in both distributions. That is, there exists some negligible function $\nu(k)$ such that:

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

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

Definition 15 (Forged messages) *Let E be an execution of* ULS *with $\mathcal{UF}$ in the UL model, and let* $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$ *be a message which $\mathcal{UF}$ delivers to some node in the network during time unit u within execution E allegedly from node N_i (outside Part (I) of the refreshment phase*[14]*). We say that* $\mathtt{msg}$ *is* forged *if*

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

(b) N_i did not send any properly certified message $\mathtt{msg}' = \langle m, i, j, u, w, \sigma', v_i^u, \mathsf{cert}_i^u\rangle$*, with the same values of m, i, j, u, w as in* $\mathtt{msg}$*, during communication round w in time unit u.*[15]

(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, $\mathsf{cert}_i^u \neq \phi$ implies that cert_i^u is a valid certificate for v_i^u.)

Definition 16 (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 which is t-operational in any time unit $u' \leq u$, it holds that $s_i^{u'}, v_i^{u'}$ and $\mathsf{cert}_i^{u'}$ are all different than ϕ.*

Let GOOD *be the set of executions which 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.

[14] Messages sent during Part (I) of the refreshment protocol are verified with the keys of the time unit $u-1$, and are considered in that time unit.

[15] Intuitively, we mean to say that N_i did not send $\mathtt{msg}$ 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.

Lemma 17 *Let E be an execution of* ULS *which is good upto time unit $u-1$, let E' be any execution of* Partial-agreement *in E upto the refreshment phases at the beginning of time unit u, and denote by S the set of nodes which are t-operational throughout the execution E'. Then, at the end of the execution E' there exists a single value x such that the local output of each node in S is either x or ϕ.*

Proof: It is enough to show that the set S satisfies the premise of Lemma 13. First, since $\mathcal{UF}$ is (t,t)-limited and $n \geq 2t+1$, then the size of S is at least $|S| \geq n-t \geq \lceil\frac{n+1}{2}\rceil$. Next, since all the nodes in S are t-operational and since every message is sent using the Disperse protocol, then by Lemma 12 every node in S receives every message that every other node in S sends to it.

Since E does not contain forged messages upto time unit $u-1$ (which includes the refreshment phase at the beginning of time unit u), then the only properly certified messages which 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 which is sent by a node in S is properly certified, and thus is accepted by all nodes in S. □

Lemma 18 *Let E be an execution of* ULS *which is good upto time unit $u-1$, and let N_i be a node which 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 were operational throughout this refreshment phase.

(b) At the conclusion of the invocation of Partial-agreement *regarding N_i's public key, the local output of all the nodes in S was equal to v_i^u, the public key that N_i sent to them.*

Proof: Recall from Definition 3 that since N_i is operational at the end of the refreshment phase in time unit u, then it has reliable links to a set S of at least $n-t$ nodes which are operational throughout this refreshment phase. Using the same arguments as in Lemma 17, this set S satisfies the premise of Lemma 13. 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 13 implies that they all end the execution of Partial-agreement with v_i^u as their local output. □

Lemma 19 (Fact 1) *For any execution $E \in$ GOOD we have*

$$\Pr\left[\textsc{al-trans}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) = E\right] = \Pr\left[\textsc{ul-trans}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x}) = E\right].$$

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

1. If $\textsc{al-trans}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) \notin \mathsf{GOOD}$ then also $\textsc{ul-trans}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r})) \notin \mathsf{GOOD}$.
2. If $\textsc{al-trans}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) \in \mathsf{GOOD}$ then $\textsc{ul-trans}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r})) = \textsc{al-trans}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r})$.

For randomness $\vec{r}$, we define $sim(\vec{r})$ to be the randomness used by $\mathcal{UF}$ and the emulated network in the execution of $\mathcal{AF}$ with the real network on randomness $\vec{r}$. From the construction it is clear that sim is one-to-one and onto. It is also clear that if $\textsc{al-trans}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r})$ is not a good execution then neither is $\textsc{ul-trans}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r}))$.

To show that if $\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) \in \mathsf{GOOD}$ then $\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x},\vec{r}) = \text{UL-TRANS}_{\mathsf{ULS},\mathcal{UF}}(k,\vec{x},sim(\vec{r}))$, we only need to show that $\mathcal{AF}$ never output "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 18. 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 $E \notin \mathsf{GOOD}$ 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 20 *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 upto 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 $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$. The set of bad executions of the first type is denoted* BAD1.

Definition 21 *An execution E is said to be a* bad execution of the second type*, if E it is not a bad execution of the first type, and there exists a time unit u such that*

(a) E is good upto time unit $u-1$.

(b) E contains a forged message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$.

(c) The first such forged message in E has $v \neq v_i^u$ (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 22 *An execution E is said to be a* bad execution of the third type*, if it is not bad of the first or second type, and there exists a time unit u such*

(a) E is good upto time unit $u-1$.

(b) E contains a forged message $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$.

(c) The first such forged message in E has $v = v_i^u$ (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.

Claim 23 *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 upto time unit 0, then for every execution which is not good there must exist a last time unit upto which it is good. Since all the ways that an execution can stop being good are covered in Definitions 20, 21 and 22, then every execution which is not good must be in either BAD1, BAD2 or BAD3. □

In Lemma 24 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, This implies that these executions have only a negligible probability. Formally, we prove

Lemma 24 *Let $\epsilon_1(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \in \mathsf{BAD1}]$. 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 $\epsilon_1(k)$.*

Proof: Let $E \in \mathsf{BAD1}$, and consider the global output of E during the refreshment phase at the beginning of time unit u. From Lemma 17 it follows that in each execution of PARTIAL-AGREEMENT there is at most one value $x \neq \phi$ which is output by t-operational nodes, and so bad event (B) will not happen.

Moreover, by Lemma 18, 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 emulated nodes send this signature to N_i over the reliable links that N_i has with them, and then N_i would not have $s_i^u = v_i^u = \mathsf{cert}_i^u = \phi$).

It follows that in any execution $E \in \mathsf{BAD1}$ there must exist a set of at least $n-t$ real nodes which log a request for a signature on v_i^u but not 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 25 *Let* $\epsilon_2(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \in \mathsf{BAD2}]$. *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* $\epsilon_2(k)$.

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 which only runs the verification algorithm of the PDS. When the scheme is started with the global verification key v_{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 w.r.t v_{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 $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$ which $\mathcal{UF}$ delivers to any emulated 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 $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$ be the first forged message in E. Using the same argument as in the proof of Lemma 24 above, there is a set of at least $n-t$ t-operational nodes which all have local output $v_i^u \neq \phi$ from the corresponding invocation of PARTIAL-AGREEMENT. It now follows from Lemma 17 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 $\mathtt{msg}$).

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 $\mathtt{msg}$ 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 26 *Let* $\epsilon_3(k) \stackrel{\text{def}}{=} \Pr[\text{AL-TRANS}_{\mathsf{ALS},\mathcal{AF}}(k,\vec{x}) \in \mathsf{BAD1}]$, *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* $\epsilon_3(k)/(n \cdot p(k))$.

Proof: $\mathcal{CF}$ is given a public key v for the scheme CS and access to a signer which has the corresponding signature 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 emulates 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 it time unit u, instead of picking the local keys for emulated node N_i using the key-generation algorithm it sets the public key $v_i^u = v$ (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 $\mathtt{msg} = \langle m, i, j, u, w, \sigma, v, \mathsf{cert}\rangle$ (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 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 occur with probability at least $\epsilon_3(k)/(n \cdot p(k))$. □

Lemmas 24, 25 and 26 yield

Corollary 27 (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 conclude the proof of Theorem 14. □

An extra property of the scheme ULS. Before concluding this section, let us highlight the following property of the constructed scheme. Recall 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 each execution in this set 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 which N_j accepts from N_i was indeed sent by N_i in this time unit.

Note that for 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 non-operational nodes is used in Section 5 to show *awareness* of the authenticator based on ULS.

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 l 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_i and 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.

Our authenticator, denoted Λ, 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.* 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 setup 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.

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.

Denote the resulting protocol $\Lambda\pi$.

Theorem 28 *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) and will not be repeated here. □

Proposition 29 *The authenticator Λ is (t, t)-aware.*

Proof: We need to show that as long as the adversary is (t, t)-limited, every non-broken node that is impersonated outputs `alert', in the same time unit. To show that, recall that a node which has $\mathsf{cert}_i^u = \phi$ always outputs `alert'. Furthermore, it follows from the extra property of scheme ULS (see the remark at the end of Section 4) that a non-broken node for which $\mathsf{cert}_i^u \neq \phi$ can not be impersonated (except with a negligible probability). □

5.1 Stronger adversaries

Before concluding, let us briefly recall a point from the Introduction regarding the properties of our scheme when the UL adversary is *not* (t,t)-limited. Although in general we cannot guarantee anything if the adversary is too strong, there is one type of stronger adversaries for which we can still prove a (weaker) awareness property. 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.

Notice that 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 will output `alert´`. Now we have two case: 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.

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.

References

[1] D. Beaver. Foundation of secure interactive computing. *Proceedings of CRYPTO'91.*

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

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

[4] M. Bellare, R. Canetti and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. STOC 98, to appear. Also available at at the Theory of Cryptography Library, http://theory.lcs.mit.edu/ tcryptol, March 1998.

[5] W. Diffie, P. C. Van Oorschot and M. J. Weiner. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography 2*, pages 107-125, 1992.

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

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

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

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

[10] R. Canetti. Modular composition of secure multiparty protocols. Manuscript, 1998.

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

[12] R. Canetti, U. Feige, O. Goldreich and M. Naor, “Adaptively Secure Computation”, 28th STOC 1996. Fuller version in MIT-LCS-TR #682, 1996.

[13] R. Canetti, R. Gennaro, A. Herzberg, D. Naor, “Proactive security: Long-term Protection against break-ins”, CryptoBytes, Vol. 3, No. 1, 1997.

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

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

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

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

[18] P. Gemmell. An introduction to threshold cryptography. In *Cryptobytes, Winter 97*, pages 7–12, 1997.

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

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

[21] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the* 19^{th} *STOC, New York City*, pages 218–229, 1987.

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

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

[24] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for the Internet. Proceedings of the IEEE Symposium on Network and Distributed System Security, 1996.

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

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

[27] S. Micali and P. Rogaway. Secure Computation. In *Proceedings of CRYPTO'91.*

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

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

[30] T. Rabin, Proactive RSA made easy. manuscript.

[31] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. *21st STOC,* 1989, pp. 73-85.

[32] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. In *22nd STOC*, pages 387–394, 1990.