Asynchronous Proactive Cryptosystems Without Agreement
(extended abstract)*

Bartosz Przydatek[1] and Reto Strobl[2]

[1] Department of Computer Science, ETH Zürich, Switzerland
[2] IBM Research, Zurich Research Laboratory, Switzerland

Abstract. In this paper, we present efficient asynchronous protocols that allow to build proactive cryptosystems secure against a mobile fail-stop adversary. Such systems distribute the power of a public-key cryptosystem among a set of servers, so that the security and functionality of the overall system is preserved against an adversary that crashes and/or eavesdrops every server repeatedly and transiently, but no more than a certain fraction of the servers at a given time. The building blocks of proactive cryptosystems — to which we present novel solutions — are protocols for *joint random secret sharing* and for *proactive secret sharing*.
The first protocol provides every server with a share of a random value unpredictable by the adversary, and the second allows to change the shared representation of a secret value. Synchronous protocols for these tasks are well-known, but the standard method for adapting them to the asynchronous model requires an asynchronous agreement sub-protocol. Our solutions are more efficient as they go without such an agreement sub-protocol. Moreover, they are the first solutions for such protocols having a bounded *worst-case* complexity, as opposed to only a bounded *average-case* complexity.

1 Introduction

Threshold cryptography addresses the task of distributing a cryptosystem among n servers such that the security and functionality of this distributed system is guaranteed even if an adversary corrupts up to t servers [2] (see [3] for a survey). Threshold cryptosystems are realized by sharing the key of the underlying cryptosystem among all servers using a $(t+1)$-out-n sharing scheme [4], and by accomplishing the cryptographic task through a distributed protocol. If this task involves the choice of secret random values, then the distribution of the task involves so-called *joint random secret sharing* (JRSS) [5], which allow the servers to jointly generate a $(t+1)$-out-n sharing of a random value unpredictable by the adversary.

Proactive cryptosystems use threshold cryptosystems as the basis, but drastically reduce the assumption concerning failures [6] (see [7] for a survey). They operate in a sequence of time periods called *phases* and tolerate a *mobile adversary*, which corrupts the servers transiently and repeatedly, and is only restricted to corrupt at most t servers during every phase. Technically, proactive cryptosystems are threshold cryptosystems

* The full version of this paper is available as an IBM Technical Report [1]

that change the representation of the shared secret key from one phase to another using *proactive secret sharing* (PSS) [8], so that the representations are independent; the old representation has to be *erased*.

The key to efficient proactivization of many public key cryptosystems for signing and encryption lies in efficient solutions for JRSS and for PSS. In the synchronous network model with broadcast channels, such solutions exist [5, 8]. Although such synchrony assumptions are justified in principle by the existence of clock synchronization and broadcast protocols, this approach may lead to rather expensive solutions in practice, for example when deployed in wide-area distributed systems with only loosely synchronized clocks. Furthermore, such systems are vulnerable to timing attacks.

These issues can be eliminated by considering an asynchronous network in the first place. However, the standard approach to building asynchronous protocols for JRSS and PSS requires an asynchronous agreement sub-protocol, which substantially contributes to the overall complexity of such solutions; see for example [9].

Contributions. In this paper, we provide the first solutions for asynchronous JRSS and for asynchronous PSS, which do not rely on an agreement sub-protocol. Avoiding agreement results in two main advantages. On one hand, we are able to bound the *worst-case* complexity of our protocols. For previous protocols, one could only bound their *average case* complexity; such protocols therefore could (at least theoretically) run forever. On the other hand, our protocols have a worst-case latency of only six rounds, whereas the best known previous solution of Cachin et al. [9] has an *expected* latency of 17 rounds (this comparison takes into account that [9] can be optimized in our model).

Our protocols tolerate a fail-stop adversary who may adaptively and repeatedly eavesdrop and crash up to t servers in every two subsequent phases, where $t < n/3$. We stress that assuming a fail-stop adversary (as opposed to a fully Byzantine adversary) does not make the problem of avoiding agreement trivial: the main reason why the standard solutions for asynchronous JRSS and PSS require agreement is the fact that a crashed server cannot be distinguished from a slow server, and this problem also occurs for a fail-stop adversary. Note that in principle our protocols can be extended to tolerate Byzantine adversaries without affecting the resilience of $t < n/3$, using known techniques for asynchronous verifiable secret sharing [9] and zero-knowledge proofs [10]. Furthermore, as shown in [11, Chapter 7], our protocols remain secure even under arbitrary composition.

The cost of our approach is a higher communication complexity. Specifically, if k is the security parameter of the system, our protocols transmit a total of $O(kn^4)$ bits across the network using $O(n^3)$ messages, whereas the (optimized) solution of Cachin et al. [9] transmits only $O(kn^3)$ bits using also $O(n^3)$ messages. However, in a practical setting, this additional overhead is of little concern as the size of n is typically very small relative to k (e.g. 10 vs. 1024).

Technically, the key to our solutions is a novel *proactive pseudorandomness* (PPR) scheme [12], with an additional property that we call *constructibility*. Such a scheme provides at every phase to every server P_i a random value pr_i which remains hidden from the adversary. Additionally, it enables the honest servers to jointly reconstruct any such value pr_i. We then build our JRSS and PSS schemes such that a server P_i derives all its random choices from its value pr_i by using it as a seed to a pseudorandom function [13]. This allows the honest servers to reproduce the steps of a (possibly) faulty server in public, instead of agreeing on a set of such servers and then excluding them from the computation (as it is done by previous work).

Related Work. As mentioned previously, Cachin *et al.* [9] implemented asynchronous proactive protocols using an agreement subprotocol as a building block, which results in a relatively high round complexity. Zhou [14] proposed to build proactive cryptosystems on a weaker notion of PSS, which can be implemented without agreement. In this weaker PSS protocol, every server computes in every phase a list of candidate shares such that one of these candidates is the fresh share of the secret. Zhou shows that this suffices to implement a proactive version of RSA signatures exploiting the fact that RSA signatures are unique in the sense that for any public key and any message, there exists only *one* signature on the given message valid under the given public key. Unfortunately, the approach of Zhou [14] cannot be applied to proactivize discrete-logarithm signature schemes such as ElGamal [15] or DSS [16], as these schemes are not unique in the above sense. The only known technique for proactivizing these signature schemes are protocols for JRSS and for PSS in the sense we introduced them before.

Organization. In the next section we introduce our system model, and recall the definitions of cryptographic tools we use in the proposed solutions. In Section 3 we give an overview of our constructions. Section 4 presents an efficient secret sharing protocol, which will be useful in our constructions. In Section 5 we present our solution for an asynchronous proactive pseudorandomness scheme. In Sections 6 and 7, we describe our solutions to asynchronous proactive secret sharing, and to asynchronous joint random secret sharing, respectively. In Section 8 we sketch how these protocols can be used to proactivize public-key signature schemes, considering Schnorr's signature scheme [17] as an example. Finally, in Section 9 we conclude the paper.

2 Asynchronous Proactive System Model

Motivation. Proactive cryptosystems are threshold cryptosystems that operate in a sequence of phases. At the beginning of every phase, the servers *refresh* the shares of the underlying threshold system such that the new shares are independent of the old shares (except for the fact that they define the same secret). This prevents an adversary from learning the shared key, assuming that she corrupts no more than t servers in every phase. Such an assumption can be justified if every phase lasts some limited amount of real time, the idea being that it takes the adversary a certain amount of real time to corrupt a server, and that corruptions are transient, i.e., do not last forever [6].

This idea maps onto a synchronous network in a straightforward way: one can define phases with respect to a common clock accessible to every server and implement refresh using a synchronous protocol [8]. The drawback of this approach is that synchronous protocols proceed in rounds, i.e., messages are sent on a clock "tick", and are received at the next "tick". This may lead to slow protocols in practice, as the duration of a communication round must account for maximal message delays and maximal shifts among local clocks of the servers. Moreover, as the security of synchronous protocols relies on the timely delivery of messages, this approach is also vulnerable to timing attacks, which are often easy to launch.

Cachin et. al [9] suggest to avoid these issues by implementing refresh using an asynchronous protocol. Such protocols are message-driven, i.e., proceed as soon as messages arrive. This allows a server to terminate a refresh and proceed with the next phase as soon as it has received enough information. Moreover, such protocols do *not* rely on upper bounds on message delays or clock shifts, i.e., they are as fast as the network. Timing attacks will only slow down such protocols, but not affect their security.

However, in a purely asynchronous network servers would not have access to a common clock for defining phases. Therefore, Cachin et al. [9] suggest to define phases *locally* to every server in terms of a single time signal, or *clock tick*, that occurs locally for a server and only indicates the start of a phase. The idea is to model systems where the time signals come from a local clock, say every day at 0:00 UTC, and where the local clocks are loosely synchronized, say they agree on which day and hour it is. Hence, the model is partially synchronous with long stretches of asynchrony. Such a setting implies an upper bound on the real time available to an adversary for corrupting servers in a certain phase, which justifies the assumption that an adversary corrupts only t servers in the same local phase [6].

The formal model of [9] does not further constrain the synchronization of phases, i.e., it leaves the scheduling of phases up to the adversary. This is to ensure that the security of a protocol does not rely on any synchrony assumptions, and hence, is not affected by timing attacks.

Network and Adversary. We adopt the basic system model from [9], which is parameterized by a security parameter k; a function $\epsilon(k)$ is called *negligible* if for all $c > 0$ there exists a k_0 such that $\epsilon(k) < \frac{1}{k^c}$ for all $k > k_0$. The network consists of n servers $P_1, \ldots, P_n$ and an adversary which are all probabilistic interactive Turing machines (PITM) [10] that run in polynomial time in k. The random tape of a server is initialized at the beginning of the computation, and we assume that the servers can *erase* information. There is also an initialization algorithm run by a trusted dealer before the system starts. On input k, n, t, and further parameters, it generates the state information used to initialize the servers.

Every server operates in a sequence of $m(k)$ *local phases*, where $m(k)$ is a polynomial. The phases are defined with respect to dedicated *input actions* of the form $(\mathsf{in}, \mathsf{clock\_tick})$, scheduled by the adversary. The local phase of a server is defined as the number of such input actions it has received.

The servers are connected by a *proactive secure asynchronous network* that allows every pair of servers to communicate authentically and privately whenever they are in the same local phase. The scheduling of the communication is determined by the adversary. Formally, we model such a network as follows. There exists a global set of messages $\mathcal{M}$, whose elements are identified by a *label* (s, r, l, τ) denoting the sender s, the receiver r, the length l of the message, and the phase τ when the message has been sent. The adversary sees the labels of all messages in $\mathcal{M}$, but not their contents. All communication is driven by the adversary, and proceeds in steps as follows. Initially, $\mathcal{M}$ is empty. At each step, the adversary performs some computation, chooses a server P_i, and selects some message $m \in \mathcal{M}$ with label (s, i, l, τ), where P_i must be currently in local phase τ. The message m is then removed from $\mathcal{M}$, and P_i is *activated* with m on its communication input tape. When activated, P_i reads the contents of its communication input tape, performs some computation, and generates one or more response messages, which it writes to its communication output tape. Then, the response messages are added to $\mathcal{M}$, and control returns to the adversary. This step is repeated arbitrarily often until the adversary halts. We view this sequence of steps as logical time, and sometimes use the phrase "at a certain point in time" to refer to such a step. Such proactive secure asynchronous networks can be implemented based on a secure co-processor [18], or on the assumption that the network itself is authentic during short periods of time, allowing the exchange of fresh communication keys [19].

We assume an *adaptive mobile fail-stop* adversary. The adversary may corrupt a server P_i at any point in time by activating it on a special input action. After such an event, she may read the entire internal state of P_i, which includes its random tape but not previously erased information. Furthermore, she may observe all messages being received, until she *leaves* the server. During such a period of time, we call a server *corrupted*; at every other point in time, a server is called *honest*. The adversary may also cause a corrupted server to stop executing a protocol. We call an adversary t*-limited* if for any phase τ, she corrupts at most t servers that are in a local phase τ or $\tau + 1$.

Protocol execution and notation. In our model, protocols are invoked by the adversary. Every protocol *instance* is identified by a unique string ID, which is chosen by the adversary when it invokes the instance. For a protocol instance ID, we model the specific input and output *actions* of a server in terms of messages of the form $(ID, \mathsf{in}, \dots)$ and $(ID, \mathsf{out}, \dots)$ that a server may receive and produce, respectively. Messages that servers send to each other over the network on behalf of an instance ID have the form $(ID, \mathsf{type}, \dots)$, where type is defined by the protocol. We call a message *associated* with a protocol instance ID if it is of the form $(ID, \dots)$.

We describe a protocol in terms of *transition rules* that are executed in parallel. Such a transition rule consists of a condition on received messages and other state variables, and of a sequence of statements to be executed in case the condition is satisfied. We define *parallel* execution of transition rules as follows. When a server is activated and the condition of one or more transition rule is satisfied, one such rule is chosen arbitrarily and the corresponding statements are executed. This is repeated until no more conditions of transition rules are satisfied. Then, the activation of the server is terminated.

A protocol instance may also invoke another protocol instance by sending it a suitable input action and obtain its output via an output action. We assume that there is an appropriate server-internal mechanism which creates the instance for the sub-protocol, delivers the input message, and passes the produced output message to the calling protocol. Furthermore, we assume that upon termination of a protocol instance, all internal variables associated with this instance are erased.

Efficiency Measures and Termination. We define the *message complexity* of a protocol instance as the number of all associated messages produced by honest servers. It is a family of random variables that depend on the adversary and on k. Similarly, the *communication complexity* of a protocol instance is defined as the bit length of all associated messages, and is also a family of such random variables. To define the *latency* (round complexity) of a protocol, we follow the approach of [20], where informally speaking the latency of an execution is the absolute duration of the execution divided by a longest message delay in this execution, where both times are as measured by an imaginary external clock. The latency of a protocol is a latency of a worst-case execution. (For details see the full version of [20], page 6.)

These quantities define a *protocol statistic* X, i.e., a family of real-valued, non-negative random variables $\{X_A(k)\}$, parameterized by the adversary A and the security parameter k, where each $X_A(k)$ is a random variable induced by running the system with A. We call a protocol statistic *uniformly bounded* if there exists a fixed polynomial $p(k)$ such that for all adversaries A, the probability $\Pr[X_A(k) > p(k)]$ is negligible.

As usual in asynchronous networks, we require *liveness* of a protocol, i.e., that "something good" eventually happens, only to the extent that the adversary delivers in every phase all associated messages among the servers that remain honest during this phase. As in [21], we define termination as the combination of liveness and an efficiency condition, which requires a protocol to have *uniformly bounded* message complexity, i.e., the number of messages produced by the protocol is independent of the adversary.

Cryptographic Assumptions. Our constructions are based on the assumption that there exists pseudo-random functions [13] defined as follows (sketch): Let $\mathcal{F}_k$ denote the set of functions from $\{0,1\}^k \to \{0,1\}^k$, and let $e \in_R Dom$ denote the process of choosing an element e uniformly at random from domain Dom. Finally, let D^f denote the execution of an algorithm D when given oracle access to f, where f is a random variable over $\mathcal{F}_k$. We say that D with oracle access distinguishes between two random variables ψ and g over $\mathcal{F}_k$ with gap $s(k)$, if $|\Pr[D^\psi(1^k) = 1] - \Pr[D^g(1^k) = 1]| = s(k)$. We say a random variable ψ over $\mathcal{F}_k$ is $s(k)$-pseudorandom, if no polynomial time in k algorithm D with oracle access distinguishes ψ from $g \in_R \mathcal{F}_k$ with gap $s(k)$.

A function family $\Psi_k = \{\psi_l\}_{l\in\{0,1\}^k}$ (with $\psi_l \in \mathcal{F}_k$) is called $s(k)$*-pseudorandom*, if the random variable ψ_l for $l \in_R \{0,1\}^k$ is $s(k)$-pseudorandom. If $s(k)$ is negligible, the collection $\{\Psi_k\}_{k\in\mathbb{N}}$ is called pseudorandom. We consider pseudorandom collections which are efficiently constructible, i.e., there exists a polynomial time algorithm that on input $l, x \in \{0,1\}^k$ outputs $\psi_l(x)$.

Pseudorandom function families can be constructed from any pseudorandom generator [13], which in turn could be constructed from any one-way function [22]. Alternatively, one could trust and use much simpler constructions based on AES or other widely available cryptographic functions.

In our protocols we make use also of *distributed* pseudorandom functions (DPRF), as introduced by Naor *et al.* [23]. In a DPRF the ability to evaluate the function is distributed among the servers, such that any authorized subset of the servers can evaluate the function, while no unauthorized subset gets any information about the function. For example, in a *threshold* DPRF the authorization to the evaluation of the functions is determined by the cardinality of the subset of the servers. In the sequel, we denote by $\Phi_k = \{\varphi_l\}_{l\in\{0,1\}^k}$ a family of efficiently constructible distributed pseudorandom functions. Moreover, we assume that if Φ_k denotes a DPRF *with threshold* κ, and if every server holds a polynomial κ-out-n share r_i of a seed r (where all r_i's are from the same domain as r) then $\varphi_r(x)$ can be efficiently computed from any set of κ values $\varphi_{r_i}(x)$ for any position $x \in \{0,1\}^k$. Threshold DPRFs with this property are also called *non-interactive*. Nielsen [24] showed how to construct efficiently such non-interactive threshold DPRFs based on the decisional Diffie-Hellman assumption [25].

3 Technical Roadmap

Hybrid Secret Sharing. A basic tool we need is a *κ-out-n hybrid secret sharing* scheme: it allows a dealer to share a secret value among all other servers, such that every server receives an *additive* n-out-n share of the secret, as well as a κ-out-n backup share of every other server's additive share $(t+1 \leq \kappa \leq n)$. Moreover, it guarantees to terminate for any server if the dealer is honest; otherwise, it either terminates for none or for all honest servers. Details of our scheme are given in Section 4.

Reconstructible Proactive Pseudorandomness (PPR). The key to our solutions for proactive secret sharing and for joint random secret sharing is a reconstructible PPR scheme. Such a scheme provides at every phase τ to every server P_i a secret value $pr_{\tau,i}$ which looks completely random to the adversary. Furthermore, any set of $n-t$ servers must be able to reconstruct the value $pr_{\tau,j}$ of any other server P_j without affecting the secrecy of the random value $pr_{\tau',j}$ computed by this server in another phase $\tau' \neq \tau$.

Our implementation assumes a trusted dealer that provides in the first phase every server P_i with a random key r_i, and with a $(n-t)$-out-n backup share r_{ji} of every other server's key r_j. The idea is to compute $pr_{\tau,i}$ as $\varphi_{r_i}(\mathtt{c})$, where $\{\varphi_l\}$ is a DPRF with threshold $(n-t)$, and $\mathtt{c}$ is some constant (pseudorandomness and constructibility of $pr_{\tau,i}$ then follows by the properties of DPRFs). This approach requires the servers to refresh in every phase their keys r_i (and shares $r_{1i}, \ldots, r_{ni}$) such that the fresh keys of honest servers are unknown to the adversary. This can be done as follows.

In a first step, P_i shares the pseudorandom value $\psi_{r_i}(\mathtt{a})$ (where $\mathtt{a}$ denotes some public constant) among all other servers using a $(n-t)$-out-n hybrid secret sharing scheme, where it derives *all* random choices using its current key r_i as a seed to a pseudorandom function. It then computes its new key r_i' as the sum of the additive shares provided by *all* these hybrid secret sharing schemes (the new shares $r_{1i}', \ldots, r_{ni}'$ are computed as the sum of all provided backup shares). To do this, P_i waits until $n-t$ servers have completed their sharing scheme as a dealer; for every other server P_j, it reveals the share r_{ji}. It can now simply wait until either P_j's sharing scheme terminates, or until it receives enough shares r_{jl} from other servers P_l to reconstruct r_j and derive the missing shares thereof; since a sharing scheme terminates either for none or for all servers, one of the two cases eventually happens.

Notice that the servers need *not* agree on whether to derive the missing shares from the sharing schemes, or from the reconstructed key r_j, as both ways provide the *same* values. Our protocol ensures that there is at least one honest server whose sharing scheme is *not* reconstructed. This ensures secrecy of the new keys r_i'.

Proactive Secret Sharing (PSS). Suppose that at the beginning of the computation, a trusted dealer shares a secret s among the servers. To prevent a mobile adversary from learning s, the servers have to compute *fresh* shares of s whenever they enter a new phase. This can be done using a proactive secret sharing scheme.

Our implementation for PSS relies on an underlying PPR scheme (initialized by the dealer). Furthermore, it assumes that the trusted dealer initially provides every server with an *additive* share of the secret s, and with a $(t+1)$-out-n *backup* share of every other server's additive share.

In an epoch τ, the servers refresh their shares of the secret by first re-sharing their additive share of s using a $(t+1)$-out-n hybrid sharing scheme; in this step, every server

P_i derives *all* its random choices by using the current random value $pr_{\tau,i}$ (provided by the PPR scheme) as a seed to a pseudorandom function.

As in the PPR scheme, every server then computes its fresh additive share of s as the sum of the additive shares provided by *all* re-sharing protocols (the backup shares are computed analogously). It therefore waits for $n-t$ re-sharing schemes to terminate, and reconstructs the remaining schemes in public. This can be done by reconstructing for every corresponding dealer P_j the random value $pr_{\tau,j}$ as well as P_j's current additive share of the secret. Reconstructing $pr_{\tau,j}$ can be done using the reconstruction mechanism of the PPR scheme, whereas P_j's additive share can be reconstructed by revealing the corresponding backup shares.

Joint Random Secret Sharing (JRSS). The goal of a JRSS protocol is to provide every server with a $(t+1)$-out-n share of a random value e unknown by the adversary. It can be executed repeatedly during the phases. Our implementation works exactly as the above protocol for refreshing a sharing, except for the following differences. In an instance with tag ID of protocol JRSS, a server P_i derives its random choices from the (pseudo)random value $\varphi_{r_i}(ID)$ (as opposed to $pr_{\tau,i} = \varphi_{r_i}(\mathsf{c})$), where r_i and $\{\varphi_l\}$ is the current key of P_i and the DPRF, respectively, used by the underlying PPR scheme. It then shares a *random* value e_i and proceeds as above. If the sharing scheme of a server P_j needs to be reconstructed, the servers reconstruct only the corresponding randomness $\varphi_{r_j}(ID)$. Adding up all backup shares provided by the sharing schemes yields the desired $(t+1)$-out-n share of the random value $e = e_1 + \cdots + e_n$.

Building Proactive Cryptosystems. Our protocols for PSS and for JRSS allow to build proactive versions of a large class of discrete logarithm-based cryptosystems without the use of expensive agreement sub-protocols. The idea is to share the key of the cryptosystem using our PSS protocol, and to accomplish the cryptographic operation using a distributed protocol. Such a protocol can be derived by combining our JRSS protocol with known techniques from threshold cryptography. We illustrate this idea in Section 8, considering Schnorr's signature scheme [17] as example.

4 Hybrid Secret Sharing

In this section, we describe the syntax and security properties of our protocol for hybrid secret sharing, $\mathsf{HybridShare}_\kappa$, which will serve as a basic tool in our subsequent constructions. A description and analysis of the protocol is given in [1].

Intuitively, our hybrid secret sharing protocol allows a dealer to share a secret s among n servers in such a way that every server P_i computes an *additive* share s_i of the secret, and a κ-out-n *backup* share s_{ji} of every other server's additive share, where $t+1 \leq \kappa \leq n$ (the idea of backing up additive shares is inspired by [26]). Our specification treats the randomness r used by the dealer as an explicit parameter, and requires that the share of every server is a deterministic function of s and r. This constructibility of the shares will be essential for our purposes.

Formally, our sharing protocol $\mathsf{HybridShare}_\kappa$ has the following syntax. Let $\mathbb{F}_q$ be an arbitrary finite field, denoting the domain of secrets. There is a distinguished server P_d called the *dealer* which *activates* an instance $ID.d$ of $\mathsf{HybridShare}_\kappa$ upon receiving an input of the form $(ID.d, \mathsf{in}, \mathsf{share}, s, r)$, where $s \in \mathbb{F}_q$ and $r \in \{0,1\}^k$; if this happens, we also say the dealer *shares* s *over* $\mathbb{F}_q$ *using randomness* r *through* $ID.d$. Every other server activates $ID.d$ upon receiving a message $(ID.d, \mathsf{in}, \mathsf{share})$. A server *terminates* $ID.d$ when it produces an output of the form $(ID.d, \mathsf{out}, \mathsf{shared}, s_i, s_{1i}, \ldots, s_{ni})$, where $s_i, s_{1i}, \ldots, s_{ni} \in \mathbb{F}_q$.

Our protocol $\mathsf{HybridShare}_\kappa$ has message complexity of $\mathcal{O}(n^2)$, communication complexity of $\mathcal{O}(kn^3)$ bits, and round complexity equal four. Furthermore, for any t-limited adversary where $t < \frac{n}{3}$, the following holds: Whenever a dealer shares a secret s over $\mathbb{F}_q$ using randomness r through an instance $ID.d$ of $\mathsf{HybridShare}_\kappa$, it holds that:

LIVENESS: If the dealer is honest throughout $ID.d$, then all honest servers terminate $ID.d$, provided all servers activate $ID.d$ in the same phase τ, and the adversary delivers all messages among servers honest during phase τ.

AGREEMENT: If one honest server terminates $ID.d$, then all honest servers terminate $ID.d$, provided all servers activate $ID.d$ in the same phase τ, and the adversary delivers all messages among servers honest during phase τ.

CORRECTNESS: The values s and r uniquely define n polynomials $f_j(x) \in \mathbb{F}_q[x]$ for $j \in [1, n]$ of degree κ, such that $s = \sum_{j=1}^{n} f_j(0)$, and the following holds: If a server P_i outputs $s_i, s_{1i}, \ldots, s_{ni}$, then $f_i(0) = s_i$ and $f_j(i) = s_{ji}$ for $j \in [1, n]$.

PRIVACY: If the dealer is honest throughout $ID.d$, and s and r are uniformly distributed in $\mathbb{F}_q$ and $\{0, 1\}^k$, respectively, then the adversary cannot guess s with probability significantly better than $1/|\mathbb{F}_q|$.

EFFICIENCY: The message complexity of $ID.d$ is uniformly bounded.

5 Asynchronous Reconstructible Proactive Pseudorandomness

In this section we give a definition for an asynchronous reconstructible PPR scheme along the lines of [12], and describe our implementation. The security proof of the scheme is contained in the full version of the paper.

5.1 Definition

Let $l(k)$ be a fixed polynomial. An *asynchronous reconstructible proactive pseudorandomness* scheme consists of a probabilistic setup algorithm σ, a proactive pseudorandomness protocol π, and a reconstruction protocol ρ. An instance of such a scheme has an associated tag ID and works as follows.

The setup algorithm σ produces the initial state information $state_{0,i}$ and the initial random value $pr_{0,i}$ of every server P_i. It is executed at the beginning of the computation by a trusted dealer. At the beginning of every phase $\tau \in [1, m(k)]$, the servers execute an instance $ID|\mathsf{ppr}.\tau$ of π to compute a fresh pseudorandom value for phase τ. The input action for server P_i carries the state information $state_{\tau-1,i}$ of the previous phase, and has the form $(ID|\mathsf{ppr}.\tau, \mathsf{in}, state_{\tau-1,i})$. The output action comprises the pseudorandom value $pr_{\tau,i}$ and the updated state information $state_{\tau,i}$. It has the form $(ID|\mathsf{ppr}.\tau, \mathsf{out}, pr_{\tau,i}, state_{\tau,i})$. If P_i does not produce an output in phase τ (which could be the case if the server was corrupted and halted in the previous phase) then its input $state_{\tau,i}$ to the subsequent instance of π is the *empty input* $\perp$.

In every phase $\tau \in [1, m(k)]$, the servers may execute an instance $ID|\mathsf{rec}_j.\tau$ of protocol ρ to reconstruct the current pseudorandom value of server P_j. The corresponding input and output actions have the form $(ID|\mathsf{rec}_j.\tau, \mathsf{in}, state_{\tau,i})$, and $(ID|\mathsf{rec}_j.\tau, \mathsf{out}, z_i)$, respectively, where $state_{\tau,i}$ denotes the current state information of P_i. We say a server *reconstructs a value* z_i *for* P_j, if it outputs a message $(ID|\mathsf{rec}_j.\tau, \mathsf{out}, z_i)$.

As in [12], we define the security requirements with respect to the following *on-line attack*: The scheme is run in the presence of a t-limited adversary for $m(k)$ phases. At every phase τ, the adversary may also instruct the servers to reconstruct the value $pr_{\tau,i}$ of *any* server. At a certain phase l (chosen adaptively by the adversary), the adversary chooses an honest server P_j whose value $pr_{\tau,j}$ is not reconstructed at that phase. She is then given a test value v, and the execution of the scheme is resumed for phases $l+1, \ldots, m(k)$. (Our definition will require that the adversary is unable to say whether v is P_j's output at phase l, or a random value.)

For an instance ID of a PPR scheme and an adversary A, let $A(ID, PR)$ denote the output of A after an on-line attack on ID, when v is indeed the output of P_j; similarly, let $A(ID, R)$ denote the corresponding output when v is a random value.

Definition 1. *Let σ, π, and ρ be given as above. We call (σ, π, ρ) a t-*resilient asynchronous reconstructible proactive pseudorandomness scheme *if for every instance ID, and every t-limited adversary A the following properties hold:*

LIVENESS: *Every server P_i honest throughout a phase $\tau \in [1, m(k)]$ terminates instance $ID|\mathsf{ppr}.\tau$ in phase τ, provided that in every phase $\tau' \in [1, \tau]$, the adversary activates each server honest throughout τ' on $ID|\mathsf{ppr}.\tau'$, and delivers all associated messages among servers honest during phase τ'. Furthermore, if every such server P_i subsequently activates $ID|\mathsf{rec}_j.\tau$ for some $j \in [1, n]$, it reconstructs some value z_i for P_j, provided the adversary delivers all associated messages among servers honest during phase τ.*

CORRECTNESS: *If a server P_j outputs $(ID, \mathsf{out}, pr_{\tau,j}, state_{\tau,j})$ in some phase $\tau \in [1, m(k)]$, and another server P_i reconstructs z_i for P_j in phase τ, then $z_i = pr_{\tau,j}$.*

PSEUDORANDOMNESS: $|\Pr[A(ID, PR) = 1] - \Pr[A(ID, R) = 1]|$ *is negligible.*

EFFICIENCY: *The message complexity of an instance of π is uniformly bounded.*

5.2 Implementation

Let $\Phi_k = \{\varphi_i\}_{i \in \{0,1\}^k}$ denote a DPRF with threshold $n - t$, and $\mathtt{a}, \mathtt{b}, \mathtt{c}$ denote distinct arbitrary constants in the domain of Φ_k. For convenience, we view elements from $\{0,1\}^k$ as elements from $\mathbb{F}_{2^k}$ (and conversely), according to some fixed bijective map from $\{0,1\}^k$ to $\mathbb{F}_{2^k}$. All computations are done over $\mathbb{F}_{2^k}$.

The Setup Algorithm σ_{ppr}. The setup algorithm provides to every server P_i a random value $r_i \in \mathbb{F}_{2^k}$, and a $(t+1)$-out-n share $r_{ji} \in \mathbb{F}_{2^k}$ of the random value of every other server. It therefore chooses n random polynomials $f_i(x) \in \mathbb{F}_{2^k}[x]$ of degree t for $i \in [1, n]$. The initial state information of a server P_i is defined as $state_{0,i} \triangleq (f_i(0), f_1(i), \ldots, f_n(i))$. The initial pseudorandom value is computed as $pr_{0,i} \leftarrow \varphi_{f_i(0)}(\mathtt{c})$.

The Reconstruction Protocol ρ_{ppr}. Let $r_i, r_{1i}, \ldots, r_{ni}$ denote P_i's local input to an instance $ID|\mathsf{rec}_j.\tau$ of protocol ρ_{ppr}. Reconstructing the pseudorandom value $pr_{\tau,j}$ of server P_j is straightforward. Every server P_i computes $pr_{\tau,ji} \leftarrow \varphi_{r_{ji}}(\mathtt{c})$, and sends it to every other server. Using the reconstruction mechanism of Φ_k, every server can compute $pr_{\tau,j}$ upon receiving $n - t$ "shares" $pr_{\tau,jm}$ from other servers P_m.

The Asynchronous Proactive Pseudorandomness Protocol π_{ppr}. Let $r_i, r_{1i}, \ldots, r_{ni}$ denote server P_i's local input $state_{\tau-1,i}$ to instance $ID|\mathsf{ppr}.\tau$ of π_{ppr}. To refresh this sharing, and to compute fresh pseudorandom values $\{pr_{\tau,i}\}$, every server P_i executes the following transition rules in parallel.

SHARE: When P_i invokes the protocol with non-empty input, it shares the pseudorandom value $\varphi_{r_i}(\mathtt{a})$ over $\mathbb{F}_{2^k}$ using randomness $\varphi_{r_i}(\mathtt{b})$ through an instance $ID|\mathsf{ppr}.\tau|\mathsf{share}.i$ of protocol $\mathsf{HybridShare}_{n-t}$.

SHARE-TERMINATION: Whenever P_i terminates a sharing protocol $ID|\mathsf{ppr}.\tau|\mathsf{share}.j$, it stores the corresponding output in the local variables $\bar{d}_{ji}, \bar{d}_{j1i}, \ldots, \bar{d}_{jni}$. If the $(n-t)$'th such sharing protocol has terminated and P_i has received non-empty input before, it sends to all servers a reveal message containing the values $\varphi_{r_{mi}}(\mathtt{a})$ and $\varphi_{r_{mi}}(\mathtt{b})$ for servers P_m whose sharing protocol *did not* terminate yet.

RECONSTRUCT: Whenever P_i receives $n-t$ values $\varphi_{r_{mi}}(\mathtt{a}), \varphi_{r_{mi}}(\mathtt{b})$ for a server P_m, it reconstructs $\varphi_{r_m}(\mathtt{a})$ and $\varphi_{r_m}(\mathtt{b})$ using the reconstruction mechanism of Φ_k. It then computes the values $\bar{d}_{mi}, \bar{d}_{m1i}, \ldots, \bar{d}_{mni}$ as the i'th share when sharing a secret $\varphi_{r_m}(\mathtt{a})$ using randomness $\varphi_{r_m}(\mathtt{b})$ according to protocol $\mathsf{HybridShare}_{n-t}$.

COMBINE: When P_i has computed values $\bar{d}_{ji}, \bar{d}_{j1i}, \ldots, \bar{d}_{jni}$ for every $j \in [1,n]$, it computes its local output values $pr_{\tau,i}$ and $state_{\tau,i} \triangleq (r'_i, r'_{1i}, \ldots, r'_{ni})$ as $r'_i \leftarrow \sum_{j=1}^{n} \bar{d}_{ji}$, $r'_{mi} \leftarrow \sum_{j=1}^{n} \bar{d}_{jmi}$ for $m \in [1,n]$, and $pr_{\tau,i} \leftarrow \varphi_{r'_i}(\mathtt{c})$.

The scheme guarantees pseudorandomness because the pseudorandom values $\varphi_{r_h}(\mathtt{a})$ and $\varphi_{r_h}(\mathtt{b})$ of at least one honest server remain hidden from the adversary. This is guaranteed because all honest servers together reveal at most $(n-t)t$ "shares" $\varphi_{r_{ij}}(\mathtt{a})$ and $\varphi_{r_{ij}}(\mathtt{b})$. But to reconstruct $\varphi_{r_i}(\mathtt{a})$ and $\varphi_{r_i}(\mathtt{b})$ of *all* $(n-t)$ honest servers, the adversary needs at least $(n-t)(n-2t) \geq (n-t)(t+1)$ such shares, as the threshold of Ψ_k is $(n-t)$.

The reason why the scheme avoids an agreement (while preserving constructibility) is the following: if an honest server P_i terminates the protocol $ID|\mathsf{ppr}.\tau|\mathsf{share}.j$ and computes the tuple $(\bar{d}_{ji}, \bar{d}_{j1i}, \ldots, \bar{d}_{jni})$, then this is *the same tuple* it would compute by first reconstructing the randomness r_j of P_j from backup shares, and then reproducing the computations of P_j in the sharing protocol $ID|\mathsf{ppr}.\tau|\mathsf{share}.j$. Hence, the servers *do not have to agree* whether to compute their share of P_j's sharing protocol by the SHARE-TERMINATION or the RECONSTRUCT transition rule, respectively, as both rules provide the *same* share. We prove the following theorem in [1].

Theorem 1. $(\sigma_{\mathsf{ppr}}, \pi_{\mathsf{ppr}}, \rho_{\mathsf{ppr}})$ *is a t-resilient asynchronous reconstructible proactive pseudorandomness scheme for $t < n/3$. It has a latency of five rounds, uses $\mathcal{O}(n^3)$ messages, and has a communication complexity of $\mathcal{O}(kn^4)$ bits.*

6 Refreshing a Sharing

In this section we define an asynchronous PSS scheme along the lines of [9], and sketch our implementation. The security proof of the scheme can be found in [1].

6.1 Definition

Let K denote the domain of possible secrets, S denote the domain of possible shares, and $l(k)$ a fixed polynomial. An *asynchronous proactive secret sharing* scheme consists of a setup algorithm σ, a proactive refresh protocol π, and a reconstruction protocol ρ. An instance of a PSS has a tag ID and works as follows.

The setup algorithm produces for each server P_i the initial state information $state_{0,i}$ and the initial share $s_{0,i} \in S$ of the secret. It is executed at the beginning of the computation by the trusted dealer. At the beginning of every phase $\tau \in [1, m(k)]$ the servers execute an instance $ID|\mathsf{ref}.\tau$ of protocol π to refresh the old share $s_{\tau-1,i}$, and to update the state $state_{\tau-1,i}$. The corresponding input and output actions of server P_i have the form $(ID|\mathsf{ref}.\tau, \mathsf{in}, s_{\tau-1,i}, state_{\tau-1,i})$ and $(ID|\mathsf{ref}.\tau, \mathsf{out}, s_{\tau,i}, state_{\tau,i})$, respectively, where $s_{\tau-1,i}$ and $state_{\tau-1,i}$ equal $\perp$ in case P_i did not produce an output in phase $\tau - 1$.

In every phase $\tau \in [1, m(k)]$, the servers may execute an instance $ID|\mathsf{rec}.\tau$ of protocol ρ to reconstruct the secret. The input and output actions for server P_i have the form $(ID|\mathsf{rec}.\tau, \mathsf{in}, s_{\tau,i})$, and $(ID|\mathsf{rec}.\tau, \mathsf{out}, z_i)$, respectively, where $s_{\tau,i}$ denotes the current share as computed by the instance $ID|\mathsf{ref}.\tau$. We say that a server *reconstructs* a value z_i, when it outputs a message $(ID|\mathsf{rec}.\tau, \mathsf{out}, z_i)$.

Definition 2. *Let σ, π, and ρ be given as above. We call (σ, π, ρ) a t-*resilient asynchronous proactive secret sharing scheme*, if for every instance ID, and every t-limited adversary the following properties hold:*

LIVENESS: *Every server P_i honest throughout a phase $\tau \in [1, m(k)]$ terminates instance $ID|\mathsf{ref}.\tau$ in phase τ, provided that in every phase $\tau' \in [1, \tau]$, the adversary activates every server honest throughout phase τ' on $ID|\mathsf{ref}.\tau'$, and delivers all associated messages among servers honest during phase τ'. Further, if every such P_i subsequently activates $ID|\mathsf{rec}.\tau$, it reconstructs some value z_i, provided the adversary delivers all associated messages among servers honest during phase τ.*

CORRECTNESS: *After initialization, there exists a fixed value $s \in K$. Moreover, if an honest server reconstructs a value z_i, then $z_i = s$.*

PRIVACY: *As long as no honest server activates an instance of ρ, the adversary cannot guess s with probability significantly better than $1/|K|$.*

EFFICIENCY: *The message complexity of π and ρ is uniformly bounded.*

We stress that the security of the sharing does not depend on the timely delivery of messages. Even if the adversary fails to deliver the messages within prescribed phase, the privacy of the shared secret is not compromised.

6.2 Implementation

Our implementation of the PSS scheme is a suitable example to illustrate how the PPR scheme introduced in the previous section can be used to avoid the need for agreement even if it seems to be inherently necessary. We therefore briefly recall the standard solution [9] for PSS that depends on agreement. Here, every server initially receives a $(t + 1)$-out-n share of the secret. To refresh the shares, every server provides every other server with a $(t + 1)$-out-n sub-share of its own share, using a suitable sharing scheme. The servers then agree on a set of at least $t+1$ servers whose re-sharing scheme terminates for all honest servers, and compute the new share as the linear combination of the received sub-shares (with Lagrange coefficients).

We follow the same approach (see Section 3), but avoid agreement by reconstructing the re-sharing schemes of the slowest (possibly crashed) servers in public. However, this approach only works if the publicly reconstructed sub-shares are *identical* to the ones which the re-sharing scheme would produce. Otherwise, the servers would again have to agree on which sub-shares to reconstruct, and which to take from the re-sharing schemes. This is where the PPR scheme comes in handy, as it allows to reconstruct the random choices made by a server when it is re-sharing its share. The technical details are given below. Let the domain of possible secrets be a field $\mathbb{F}_q$ where $q \leq 2^k$. All computations are in $\mathbb{F}_q$ or $\mathbb{F}_{2^k}$, as is clear from the context.

The Setup Algorithm σ_{pss}**.** The setup algorithm provides every server with an additive share s_i of a randomly chosen secret, and with a $(t+1)$-out-n share s_{ji} of every other server's additive share. It therefore chooses n random polynomials $f_i(x) \in \mathbb{Z}_q[x]$ of degree t for $i \in [1, n]$ (the secret is defined as $s = \sum_{i=1}^{n} s_i$). The initial share of server P_i is defined as $s_{0,i} \triangleq (s_i, s_{1i}, \ldots, s_{ni})$, where $s_i = f_i(0)$ and $s_{ji} = f_j(i)$.

Additionally, the setup algorithm provides every server with the initial state information needed to initialize a PPR scheme. It therefore runs the setup algorithm σ_{ppr}, and computes the initial state information $state_{0,i}$ of server P_i as the tuple $(state^{\mathsf{ppr}}_{0,i}, pr_{0,i})$

The Reconstruction Protocol ρ_{pss}**.** The reconstruction protocol is straight forward. Every server P_i sends its input $s_{\tau,i} \triangleq (s_i, s_{1i}, \ldots, s_{ni})$ to every other server. Upon receiving $t+1$ such values the server interpolates all missing shares s_j from the received sub-shares s_{ji} by Lagrange interpolation, and computes the secret as $s = \sum_{j=1}^{n} s_j$.

The Refresh Protocol π_{pss}**.** Let $(s_i, s_{1i}, \ldots, s_{ni})$ and $(state^{\mathsf{ppr}}_{\tau-1,i}, pr_{\tau-1,i})$ denote server P_i's local input $s_{\tau-1,i}$ and $state_{\tau-1,i}$, respectively, to instance $ID|\mathsf{ref}.\tau$. To compute a fresh share $(s'_i, s'_{1i}, \ldots, s'_{ni})$ and updated state information $(state^{\mathsf{ppr}}_{\tau,i}, pr_{\tau,i})$, every server P_i executes the following transition rules in parallel:

SHARE: When P_i invokes the protocol, it activates an instance $ID|\mathsf{ppr}.\tau$ of protocol π_{ppr} with input $state^{\mathsf{ppr}}_{\tau-1,i}$ to compute $(state^{\mathsf{ppr}}_{\tau,i}, pr_{\tau,i})$. Furthermore, if P_i received non-empty input, it shares its share s_i over $\mathbb{F}_q$ using randomness $pr_{\tau-1,i}$ through an instance $ID|\mathsf{ref}.\tau|\mathsf{share}.i$ of protocol $\mathsf{HybridShare}_{t+1}$.

SHARE-TERMINATION: Whenever P_i terminates an instance $ID|\mathsf{ref}.\tau|\mathsf{share}.j$ of a sharing protocol , it stores the corresponding output in the local variables $\bar{e}_{ji}$, $\bar{e}_{j1i}, \ldots, \bar{e}_{jni}$. If for $n-t$ servers P_j the corresponding protocols $ID|\mathsf{ref}.\tau|\mathsf{share}.j$ have terminated, it sends the indices of all servers whose sharing protocol *did not* terminate yet to every other server in a missing message.

REVEAL: If for some index m, P_i receives $(n-t)$ missing messages from other servers containing this index and has received non-empty input before, it sends a reveal message to every other server containing the backup share s_{mi} and the index m. Next, it activates the instance $ID|\mathsf{rec}_m.\tau$ of protocol ρ_{ppr} with input $state^{\mathsf{ppr}}_{\tau-1,i}$ to reconstruct the randomness $pr_{\tau-1,m}$ of P_m.

RECONSTRUCT: Whenever P_i receives $(t+1)$ reveal messages for the same index m *and* reconstructs the value $pr_{\tau-1,m}$ for P_m, it computes the share s_m from the received backup shares by Lagrange interpolation. It then computes the tuple $(\bar{e}_{mi}$, $\bar{e}_{m1i}, \ldots, \bar{e}_{mni})$ as the i'th share when sharing s_m using randomness $pr_{\tau-1,m}$.

COMBINE: When P_i has computed values $(\bar{e}_{mi}, \bar{e}_{m1i}, \ldots, \bar{e}_{mni})$ for all $m \in [1, n]$, it computes the new share $(s'_i, s'_{1i}, \ldots, s'_{ni})$ as follows: $s'_i \leftarrow \sum_{j=1}^{n} \bar{e}_{ji}$, $s'_{mi} \leftarrow \sum_{j=1}^{n} \bar{e}_{jmi}$ for $m \in [1, n]$.

Notice that the protocol has the same message flow as the pseudorandomness protocol π_{ppr}, except for the additional missing messages. They ensure the secrecy of the share s_h of at least one honest server P_h, and are needed because the servers hold a $(t+1)$-out-n hybrid sharing of the secret s. We remark that for refreshing a $(n-t)$-out-n hybrid sharing, the servers could omit waiting for $t+1$ such messages, and could execute the REVEAL rule directly at the end of the SHARE-TERMINATION rule. This would save one communication round. The proof of the following theorem can be found in [1].

Theorem 2. *The tuple $(\sigma_{\mathsf{pss}}, \pi_{\mathsf{pss}}, \rho_{\mathsf{pss}})$ is a t-resilient asynchronous proactive secret sharing scheme for $t < n/3$. The refresh protocol π_{pss} uses $\mathcal{O}(n^3)$ messages, has latency of six rounds and communication complexity of $\mathcal{O}(kn^4)$.*

7 Asynchronous Proactive Joint Random Secret Sharing

The goal of an asynchronous proactive joint random secret sharing scheme is to enable the servers to repeatedly generate $(t+1)$-out-n sharings of random values, such that the random values remain hidden from the adversary. Due to lack of space, we only sketch the definition and implementation.

Definition. An asynchronous proactive joint random secret sharing (JRSS) scheme consists of a setup algorithm σ, a proactive update protocol π, a joint random secret sharing protocol γ, and a reconstruction protocol ρ. An instance of such a scheme has a tag ID and works as follows.

At the beginning of the computation, a trusted dealer executes the setup algorithm σ and provides every server with its initial state information $state_{0,i}$. At the beginning of every phase $\tau \in [1, m(k)]$, the servers execute protocol π to update the state information $\{state_{\tau-1,i}\}$. During every phase $\tau \in [1, m(k)]$, the servers can repeatedly execute protocol γ to generate a sharing of a random value z_c in a domain K. Every such instance has a unique tag $ID|\mathsf{gen}_c$. For every server P_i, it takes the current state information $state_{\tau,i}$ as input, and produces as output a share $s_{c,i}$ of the random value z_c. These shares may serve as input to the reconstruction protocol ρ with tag $ID|\mathsf{rec}_c$, which produces for every server P_i a value $z_{c,i}$ as output.

For a JRSS scheme to be secure, we require that when the first server completes an instance $ID|\mathsf{gen}_c$, there is a fixed value z_c such that the following holds: (Correctness) If a server P_i terminates $ID|\mathsf{rec}_c$ and outputs $z_{c,i}$, then $z_{c,i} = z_c$. Furthermore, (Privacy) as long as no honest server activates $ID|\mathsf{rec}_c$, the adversary cannot guess z_c with probability significantly better than $1/|K|$.

Implementation. Our implementation builds on our PPR scheme $(\sigma_{\mathsf{ppr}}, \pi_{\mathsf{ppr}}, \rho_{\mathsf{ppr}})$. Let $\Phi_k = \{\varphi_i\}$ denote the DPRF family used by the PPR scheme, $\mathtt{a}$ and $\mathtt{b}$ denote two distinct constants, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ denote a collision resistant hash function (it is well-known how to construct such functions from standard computational assumptions such as the hardness of the discrete-logarithm problem).

The state information $\{state_{\tau,i}\}$ of our JRSS scheme comprises only the state information of our PPR scheme, i.e., $state_{\tau,i} \triangleq (r_i, r_{1i}, \ldots, r_{ni})$. Protocols σ_{jrss} and π_{jrss} for setting up and refreshing this state, respectively, consist only of calling the protocols σ_{ppr} and π_{ppr}. The protocol γ_{jrss} for generating sharings of random values in $\{0,1\}^k$

works as follows. Given input $state_{\tau,i} \triangleq (r_i, r_{1i}, \ldots, r_{ni})$ to an instance $ID|\mathsf{gen}_c$ of γ_{jrss}, every server P_i performs the following steps (all computations are done in $\mathbb{F}_{2^k}$).

SHARE: When P_i invokes the protocol with non-empty input, it shares $\varphi_{r_i}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{a}))$ over $\mathbb{F}_{2^k}$ through an instance of protocol $\mathsf{HybridShare}_{t+1}$ with tag $ID|\mathsf{gen}_c|\mathsf{share}.i$ using randomness $\varphi_{r_i}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{b}))$.

SHARE-TERMINATION: Whenever P_i terminates a sharing protocol $ID|\mathsf{gen}_c|\mathsf{share}.j$, it stores the corresponding output in local variables $\bar{e}_{ji}, \bar{e}_{j1i}, \ldots, \bar{e}_{jni}$. Once $n-t$ sharing protocols have terminated and P_i has received non-empty input before, it sends to all servers a reveal message containing values $\varphi_{r_{mi}}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{a}))$ and $\varphi_{r_{mi}}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{b}))$ for servers P_m whose sharing protocol *did not* terminate yet.

RECONSTRUCT: Upon receiving $n-t$ reveal messages for the same index m, P_i reconstructs values $\varphi_{r_m}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{a}))$ and $\varphi_{r_m}(\mathcal{H}(ID|\mathsf{gen}_c|\mathtt{b}))$ (using the threshold evaluation property of Φ_k) and derives the missing sub-share $\bar{e}_{mi}, \bar{e}_{m1i}, \ldots, \bar{e}_{mni}$.

COMBINE: When P_i has computed values $(\bar{e}_{mi}, \bar{e}_{m1i}, \ldots, \bar{e}_{mni})$ for every $m \in [1,n]$, it computes $s_{c,i} \triangleq (s_i,\ s_{1i}, \ldots, s_{ni})$ as follows: $s_i \leftarrow \sum_{j=1}^{n} \bar{e}_{ji}$, $s_{mi} \leftarrow \sum_{j=1}^{n} \bar{e}_{jmi}$ for $m \in [1,n]$.

The shared secret value z_c is never reconstructed but equals $\sum_{i=1}^{n} s_i$. The protocol has a latency of five rounds, a message complexity of $O(n^3)$, and a communication complexity of $O(kn^4)$ bits.

An instance $ID|\mathsf{rec}_c$ of the reconstruction protocol ρ_{jrss} works as follows. Every server i sends its share $s_{c,i} \triangleq (s_i, s_{1i}, \ldots, s_{ni})$ — which it receives as input — to every other server. Upon receiving $t+1$ such values, P_i derives all values s_j from the received sub-shares s_{jm} by Lagrange interpolation and computes the secret as $z_c = \sum_{j=1}^{n} s_j$.

8 A Simple Proactive Secure Signature Scheme

Our protocols for PSS and JRSS can be used to proactivize a large class of discrete-logarithm based public-key cryptosystems for signing and encryption. In this section, we sketch how this can be done considering Schnorr's signature scheme as an example.

Let p denote a large prime, and $\langle g \rangle$ denote a multiplicative subgroup of $\mathbb{Z}_p^*$ of prime order q such that $q|p-1$. In the regular centralized Schnorr signature scheme, the secret key x of the signer is a random element from $\mathbb{Z}_q$, and the public key is $y = g^x$. To sign a message $m \in \{0,1\}^*$, the signer picks a random number $r \in \mathbb{Z}_q$, and computes the signature (ρ, σ) as $\rho \leftarrow g^r \mod p$ and $\sigma \leftarrow r + \mathcal{H}(m||\rho)x \mod q$. A signature (ρ, σ) on a message m can then be verified by checking that $g^\sigma = \rho y^{\mathcal{H}(m||\rho)} \mod p$.

In a proactive signature scheme, the power to sign a message is distributed among the servers such that in every epoch, only a set of at least $t+1$ servers can generate valid signatures, whereas any smaller set can neither compute a signature nor prevent the overall system from operating correctly. For a formal treatment of proactive signature schemes we refer to [12].

Proactivizing Schnorr's signature scheme in the above sense can be done as follows. First, a trusted dealer chooses the values p, q, g, x as in the standard Schnorr scheme, and initializes a PSS scheme with a sharing of x. It also initializes a JRSS scheme, and announces the public parameters p, q, g and y. To compute a signature (ρ, σ) on a message m, every server i performs the following steps:

generate $\rho = g^r$**:**

(1) Use the underlying JRSS scheme to compute a $(t+1)$-out-n share r_i of a random value $r \in \mathbb{Z}_q$.
(2) Reveal the value $\rho_i = g^{r_i} \mod p$ to all other servers.
(3) Upon receiving $t+1$ values ρ_j, compute ρ from the values ρ_j by using Lagrange interpolation in the exponent, i.e., $\rho \leftarrow \prod_{i \in Q} \rho_j^{\lambda_j} \mod p$. Here, Q denotes the indices of the received values ρ_i, and λ_i the Lagrange interpolation coefficient for the set Q and position 0.

generate $\sigma = r + \mathcal{H}(m||\rho)x$**:**

(1) Reveal the value $\sigma_i = r_i + \mathcal{H}(m||\rho)x_i \mod q$ to all other servers; here, x_i denotes server i's current share of x as computed by the underlying PSS scheme.
(2) Upon receiving $t+1$ values σ_j, compute σ by using Lagrange interpolation, i.e., $\sigma \leftarrow \sum_{j \in S} \lambda_j \sigma_j \mod q$. Here, S denotes the indices of the received values σ_j, and λ_j the Lagrange coefficients for the set S and position 0.

Verification of the computed signature can be done exactly as in the centralized Schnorr scheme. One can show that this proactive signature scheme is as secure as the centralized Schnorr scheme in the following sense: If there exists a t-limited mobile adversary against the proactive signature scheme that can forge a signature (under an adaptively chosen message attack), then there exists an adversary against the centralized Schnorr scheme that can forge signatures (under an adaptively chosen message attack).

Proactivizing other discrete-logarithm signature schemes such as ElGamal [15] or DSS [16] can be done in a similar way (to solve the inversion problem that occurs in DSS, one can use the approach of [27]).

9 Conclusions and Open Problems

In this paper, we have presented the first asynchronous schemes for proactive secret sharing and proactive joint random secret sharing with a bounded *worst case* complexity. Moreover, our solutions run three times faster (in terms of latency) than the best known previous solutions.

The technical novelty of our schemes is that they do not rely on an agreement subprotocol. The fact that agreement can be avoided is surprising on its own, as all known previous techniques for implementing such schemes require the servers to have at some point a common view of which servers have been crashed.

A natural open problem is to enhance our techniques to tolerate a Byzantine adversary. Here, the main difficulty lies in designing a *verifiable* version of our hybrid secret sharing scheme. In such a scheme, the dealer must be committed to a random value (of the same size as the secret), such that every server can verify that the dealer has indeed computed the shares by using this random value as a seed to a pseudorandom function. In principle, this can be done using the technique of general zero-knowledge proofs [10]. We suggest it as an open research problem to construct a pseudorandom function together with *efficient* zero-knowledge proofs for this task.

References

1. Przydatek, B., Strobl, R.: Asynchronous proactive cryptosystems without agreement. Technical Report RZ 3551, IBM Research (2004)

2. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Proc. CRYPTO '87. (1987)
3. Desmedt, Y.: Threshold cryptography. European Transactions on Telecommunications **5** (1994) 449–457
4. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979) 612–613
5. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure key generation for discrete-log based cryptosystems. In: Proc. EUROCRYPT '99. (1999) 295–310
6. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks. In: Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC). (1991) 51–59
7. Canetti, R., Gennaro, R., Herzberg, A., Naor, D.: Proactive security: Long-term protection against break-ins. RSA Laboratories' CryptoBytes **3** (1997)
8. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or how to cope with perpetual leakage. In: Proc. CRYPTO '95. (1995) 339–352
9. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Aysnchronous verifiable secret sharing and proactive cryptosystems. In: Proc. 9th ACM CCS. (2002)
10. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proc. 17th ACM STOC. (1985) 291–304
11. Strobl, R.: Distributed Cryptographic Protocols for Asynchronous Networks with Universal Composability. PhD thesis, ETH (2004)
12. Canetti, R., Herzberg, A.: Maintaining security in the presence of transient faults. In: Proc. CRYPTO '94. (1994) 425–438
13. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33** (1986) 792–807
14. Zhou, L.: Towards fault-tolerant and secure on-line services. PhD thesis, Cornell Univ. (2001)
15. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Info. Theory **IT 31** (1985)
16. National Institute for Standards, Technology: Digital signature standard (DSS). Technical Report 169 (1991)
17. Schnorr, C.P.: Efficient signature generation by smart cards. J. of Cryptology **4** (1991) 161–174
18. Backes, M., Cachin, C., Strobl, R.: Proactive secure message transmission in asynchronous networks. In: Proc. 21th ACM PODC. (2003)
19. Canetti, R., Halevi, S., Herzberg, A.: Maintaining authenticated communication in the presence of break-ins. J. of Cryptology **13** (2000) 61–106
20. Canetti, R., Rabin, T.: Fast asynchronous Byzantine agreement with optimal resilience. In: Proc. 25th ACM STOC. (1993) full version at www.research.ibm.com/security/cr-ba.ps.
21. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols (extended abstract). In: Proc. CRYPTO 01. (2001) 524–541
22. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM Journal on Computing **28** (1999) 1364–1396
23. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Proc. EUROCRYPT '99. (1999) 327–346
24. Nielsen, J.B.: A threshold pseudorandom function construction and its applications. In: Proc. CRYPTO '02. (2002) 401–416
25. Boneh, D.: The decision Diffie-Hellman problem. In: Third Algorithmic Number Theory Symposium. Volume 1423 of LNCS. (1998) 48–63
26. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Proc. CRYPTO '98. (1998)
27. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Proc. CRYPTO '99. (1999)