# Distributed Key Generation for the Internet*

Aniket Kate  Ian Goldberg
David R. Cheriton School of Computer Science
University of Waterloo, ON, Canada
{akate,iang}@cs.uwaterloo.ca

**Abstract**

Although distributed key generation (DKG) has been studied for some time, it has never been examined outside of the synchronous setting. We present the first realistic DKG architecture for use over the Internet. We propose a practical system model and define an efficient verifiable secret sharing scheme in it. We observe the necessity of Byzantine agreement for asynchronous DKG and analyze the difficulty of using a randomized protocol for it. Using our verifiable secret sharing scheme and a leader-based agreement protocol, we then design a DKG protocol for public-key cryptography. Finally, along with traditional proactive security, we also introduce group modification primitives in our system.

## 1 Introduction

A distributed key generation (DKG) protocol is a fundamental building block of both symmetric and asymmetric threshold cryptography. In essence, an $(n, t)$-DKG protocol [Ped91] allows a set of $n$ nodes to collectively generate a secret with its *shares* spread over the nodes such that any subset of size greater than a threshold $t$ can reveal or use the shared secret, while smaller subsets do not have any knowledge about it. Unlike secret sharing [Sha79, Bla79], where a *dealer* generates a secret and distributes its shares among the nodes, DKG requires no trusted party.

**Threshold Cryptography.** In symmetric-key cryptography, DKGs are used to design distributed key distribution centres [NPR99]. Here, a set of $n$ servers jointly realize the function of a conference-key distribution centre, which generates and provides conference keys to the clients.In public-key cryptography (PKC), they are essential for dealerless threshold public-key encryption and signature schemes [Des94] and for truly distributed private-key generation in identity-based cryptography (IBC) [BF01]. In a threshold public-key encryption scheme, a private key is distributed among a group such that, given a ciphertext, more than a threshold number of them have to combine their decrypted shares to find the plaintext message. On the other hand, in a threshold signature scheme, the signing key is distributed among a group such that more than a threshold number of them has to combine their partial signatures to sign a message. In threshold encryption and signature schemes, DKG tackles the problem of *single point of failure*. In IBC, it also mitigates the *key escrow* issue. Here, it is impractical or impossible to trust and rely on a single entity, the *private-key generator* (PKG), to generate and distribute private keys to IBC clients. A distributed PKG becomes important when IBC is used in practical systems, outside the usual organizational settings, such as key distribution in ad-hoc networks [KKA03] or pairing-based onion routing [KZG07]. It is also an important primitive in distributed pseudo-random functions [NPR99], which are useful in designing distributed coin tossing algorithms [CKS00] and random oracles [Nie02].

As a whole, numerous applications based on DKG have been proposed (see [GJKR07] and references therein). However, most of them assume a synchronous communication model or a broadcast channel. The systems issues to be considered while realizing DKGs over the Internet have largely been ignored and there is no implementation available yet. This need for a practical DKG forms the motivation of this work.

[*] This is the full version of our ICDCS 2009 paper. Revised June 2009.

### Verifiable Secret Sharing—VSS

In secret sharing, clients need to verify a consistent dealing (integrity) to prevent malicious behaviour by the dealer. A scheme with such a verifiability property is known as *verifiable secret sharing* [CGMA85]. Feldman [Fel87] developed the first efficient and non-interactive VSS protocol. He used a commitment with computational security and unconditional share integrity to achieve it. Pedersen presented another commitment [Ped91] with unconditional security but computational integrity. In computational PKC, with adversarial access to the public key, unconditional security for the secret is impossible. Further, an additional round of communication is required at start time when using Pedersen's scheme in order to randomly select generators $g$ and $h$. Consequently, with simplicity and efficiency, Feldman's commitments form the basis for many VSSs, including ours.

**Proactive VSS.** The most common attacks on security mechanisms are system attacks, where the system's cryptographic keys are directly exposed, rather than cryptanalytic attacks. Due to the endless supply of security flaws in almost all existing software, these system attacks are often easy to implement. Threshold cryptography enhances security against system break-ins, but its effect is limited. Given sufficient time, a *mobile attacker* can break into system nodes one by one (*gradual break-in*) and eventually compromise the security of the whole system [OY91]. Proactive secret sharing [HJKY95], which combines distributed trust with periodic share renewal, protects a system against these gradual break-ins. Here, the system's time is divided into *phases*. At the start of each phase, nodes' secret shares are renewed such that new shares are independent of previous ones, except for the fact that they interpolate to the same secret key. With an assumption that the adversary may corrupt at most $t$ nodes in each phase, the system now becomes secure.

**Asynchronous VSS.** Although the literature for VSS has been vast, asynchronous VSS has not yet received the required attention. Canetti and Rabin [CR93] developed the first complete VSS scheme with unconditional security in the *asynchronous communication model* having no bounds on message transfer delays or processor speeds. However, this scheme and its successors [ADH08, PCR08], due to their $\Omega(n^5)$ *communication complexities* (bit length of messages transferred), are prohibitively expensive for any realistic use. Compromising the unconditional security assumption, Cachin et al. (AVSS) [CKAS02], Zhou et al. (APSS) [ZSvR05], and more recently Schultz et al. (MPSS) [SLL08] suggested more practical asynchronous VSS schemes. APSS severely restricts $t$ with its $\Omega\left(\binom{n}{t}\right)$ *message complexity* (number of messages transferred), and is thus very ineffective in general. The bivariate polynomial based AVSS and univariate polynomial based MPSS have the same reasonable communication complexity of $O(n^3)$. However, security is preserved in MPSS only when sets of nodes used in two consecutive phases are disjoint; this is not ideal in many scenarios. On other hand, AVSS assimilates a bivariate polynomial into Bracha's reliable broadcast [Bra84] and can provide complete flexibility with the sets used without hampering the security. In an asynchronous VSS protocol with reliability guarantees, any two nodes need to verify the dealer's commitment of size $\Omega(n)$ with each other to achieve consistency; thus, a protocol with $o(n^3)$ communication complexity does not seem to be possible and AVSS, with optimal communication complexity, forms the basis for our VSS.

### Contributions

In this paper, we design the first practical DKG protocol for use over the Internet.

- As our first contribution, we define a realistic system model over the Internet (§2). We combine the standard Byzantine adversary with crash-recovery and network failures in an asynchronous setting. We analyze the asynchronous versus partially synchronous dichotomy for the Internet and justify the choice of treating crashes and network failures separately.

- We present a VSS scheme (HybridVSS) that works in our system model (§3), investigate the necessity of an agreement scheme for asynchronous DKG, and define a practical DKG protocol (§4). We use a leader-based agreement scheme in our DKG, as we observe a few pragmatic and efficiency related issues with the usually suggested randomized agreement schemes.

- Along with proactive security (§5), observing the importance of group modifications for a long-term system sustainability, we also devise protocols for group modification agreement, node addition, node removal and threshold and crash-limit modification (§6).

- Finally, we touch upon the system design and implementation that we are working on anddiscuss the system's resilience against denial-of-service (DoS) attacks and Sybil attacks (§7).

# 2 Assumptions and System Model

In this section, we discuss the assumptions and the system model for our protocols, giving special attention to their practicality over the Internet.

## 2.1 Communication Model

Our DKG protocol should be deployable over the Internet. The expected message-transfer delay and the expected clock offset there (a few seconds, in general) is significantly smaller than the required timespan of a system phase (a few days). With such an enormous difference, a failure of the network to deliver a message within a fixed time bound can be treated as a failure of the sender; this may lead to a retransmission of the message after appropriate timeout signals. As this is possible without any significant loss in the synchrony of the system, the asynchronous communication assumption seems to be unnecessarily pessimistic here. It is tempting to treat the Internet as a *partially synchronous network* (bounded message delivery delays and processor speeds, but the bounds are unknown and eventual [DLS88]) and develop more efficient protocols using well-known message delivery time bounds and system run-time assumptions.

However, deciding these time bounds correctly is a difficult problem to solve. Further, even if it is possible to determine tight bounds between the optimistic and pessimistic cases, there is a considerable difference between the selected time bounds and the usual computation and communication time. Protocols explicitly based on synchronous or partially synchronous assumptions invariably use these time bounds in their definitions, while those based on the asynchronous assumption solely use numbers and types of messages. A real-world adversary, with knowledge of any time bounds used, can always slow down the protocols by delaying its messages to the verge of the time bounds. In asynchronous protocols, although it is assumed that the adversary manages the communication channels and can delay messages as it wishes, a real-world adversary cannot control communication channels for all the honest nodes. It is practical to assume that network links between most of the honest nodes are perfect. Consequently, even if the adversary delays its messages, an asynchronous protocol completes without any delay with honest nodes communicating promptly. Thus, the asynchrony assumption may increase message complexity or the *latency degree* (number of communication rounds), but in practice does not increase the actual execution time. Observing this, we use the asynchronous communication assumption for our protocols.

### Weak Synchrony (only for liveness)

For *liveness* (the protocol eventually terminates), but not *safety* (the protocol does not fail or produce incorrect results), we need a weak synchrony assumption. Otherwise, we could implement consensus in an asynchronous system, which is impossible [FLP85]. We use a weak synchrony assumption by Castro and Liskov [CL02] to achieve liveness. Let $delay(t)$ be the time between the moment $t$ when a message is sent for the first time and the moment when it is received by its destination. The sender keeps retransmitting the message until it is received correctly. We assume that $delay(t)$ *does not grow faster than* $t$ *indefinitely.* Assuming that network faults are eventually repaired and DoS attacks eventually stop, this assumption seems to be valid in practice. It is also strictly weaker than the partially synchronous communication assumption.

## 2.2 Byzantine Adversary, Crash-Recoveries and Link Failures

Most of the distributed computing protocols in the literature assume a $t$-limited *Byzantine adversary*, who compromises up to $t$ out of $n$ system nodes and makes them behave arbitrarily. We aim at proactive security for our protocols, where the $t$-limited mobile Byzantine adversary can change its choice of $t$ nodes as time progresses. Here, a node compromised during a phase remains unused, after recovery, for the remainder of that phase as its share is already compromised. Any intra-phase share modification for a recovered node leads to intra-phase share modification to all the nodes, which is unacceptable in general.

This does not model failures over the Internet in the best way. Other than malicious attacks leading to compromise, some nodes (say $f$ of them) may just crash silently without showing arbitrary behaviours or get disconnected from the rest of the network due to network failures or partitioning. Importantly, secrets at these $f$ nodes are not available to the adversary and modelling them as Byzantine failures not only leads to sub-optimal resilience of $n \geq 3(t+f)+1$ instead of $n \geq 3t+2f+1$, but it also increases the communication complexity with added security requirements ($t+f$ instead of $t$). Keeping such nodes inactive, after their recovery, until the start of next phase is not ideal. This prompts us to use a *hybrid model*.

Our system adopts the hybrid model by Backes and Cachin [BC03], but with a modification to accommodate broken links. From any honest node's perspective, a crashed node behaves similarly to a node whose link with it is broken and we model link failures in the form of crashes. For every broken link between two nodes, we assume that at least one of two nodes is among the list of currently crashed nodes.[1] Further, all non-Byzantine nodes may crash and recover repeatedly with a maximum of $f$ crashed nodes at any instant and a recovering honest node recovers from a well-defined state using, for example, a read-only memory. We also assume that the adversary delivers all the messages between two uncrashed nodes. We drop the requirement of proactive security at this point and pick it back up in §5.

Formally, we consider an asynchronous network of $n \geq 3t+2f+1$ nodes $P_1, \ldots, P_n$ of which the adversary may corrupt up to $t$ nodes during its existence and may crash another $f$ nodes at any time. For $f=0$, $3t+1$ nodes are required as a differentiation between slow honest nodes and Byzantine nodes is not possible in an asynchronous network, while for $t=0$, $2f+1$ nodes are mandatory to achieve consistency. At least $n-t-f$ nodes, which are not in the crashed state at the end of a protocol, are termed *finally up* nodes.

## 2.3 Complexity and Cryptographic Assumptions

Our adversary is computationally bounded with a security parameter $\kappa$. A function $\epsilon(\cdot)$ is called *negligible* if for all $c > 0$ there exists a $\kappa_0$ such that $\epsilon(k) < 1/\kappa^c$ for all $\kappa > \kappa_0$.

**Complexity Assumption.** An unbounded number of crashes can cause the protocol execution time to be unbounded. We restrict the adversary by function $d(\kappa)$ that represents the maximum number of crashes that the adversary is allowed to perform during its lifetime.

Work done by honest parties can be measured by a *protocol statistic* $X$, which is a family of real-valued non-negative random variables $\{X_A(\kappa)\}$, parametrized by adversary $A$, where each $X_A(\kappa)$ is a random variable induced by running the system with $A$. A protocol statistic $X$ is called *uniformly bounded* if there exists a fixed polynomial $T(\kappa)$ such that for all adversaries $A$, there is a negligible function $\epsilon_A$, such that for all $k \geq 0$, $Pr[X_A(\kappa) > T(\kappa)] \leq \epsilon_A(\kappa)$. As we consider a computationally bounded adversary, we aim at polynomially bounded system execution space and time and bounding protocol complexities by a polynomial in the adversary's running time.

For crash-recovery situations, Backes and Cachin introduce the notion of *d-uniformly bounded statistics* [BC03, Def. 1]. Here, a bounded protocol statistic $X$ is considered to be $d$-uniformly bounded (by $T_1$ and $T_2$) for a function $d(\kappa)$ if there exist two fixed polynomials $T_1$ and $T_2$ such that for all adversaries $A$, there exists a negligible function $\epsilon_A(\kappa)$ such that for all $\kappa \geq 0$, $Pr[X_A(\kappa) > d(\kappa)T_1(\kappa) + T_2(\kappa)] \leq \epsilon_A(\kappa)$. In order words, the complexity of the protocol is uniformly bounded if no crash occurs (which is ensured by $T_2$), and the computational overhead caused by each crash is also uniformly bounded (ensured by $T_1$). Similar to [BC03], we expect that the communication complexity of a protocol is $d$-uniformly bounded for some polynomial $d$.

**Cryptographic Assumptions.** The infeasibility of the adversary to compute discrete logarithms modulo large primes forms our main cryptographic assumption. We consider a prime $p$ such that there exists a $\kappa$-bit prime $q$ and $q|(p-1)$. Let $\mathbb{G}$ be a multiplicative subgroup of $\mathbb{Z}_p^*$ of order $q$ and let $g \in \mathbb{G}$ be a generator. For every probabilistic polynomial time algorithm $\mathcal{A}$ and $x \in_R [1, q]$, the probability $Pr(\mathcal{A}(p, q, g, g^x) = x)$ is negligible.

The adversary is also *static* and *rushing*. It has to choose its $t$ compromisable nodes before a protocol run.[2] However, it can wait for the messages of the uncorrupted players to be transmitted, then decide on its

[1] A node that is crashed means that *some* of its links are down, not necessarily that they *all* are.

[2] As we use Feldman's VSS, we do not prove security against an adaptive adversary. However, as claimed by Feldman [Fel87, Sec. 9.3], although the use of simulation-based security proof did not work out for an adaptive adversary, the VSS scheme does

computation and communication for that round, and still get its messages delivered to the honest parties on time.

We use a PKI infrastructure in the form of a PKI hierarchy with an external certifying authority (CA) to achieve authenticated and confidential communication with TLS links, and message authentication with any digital signature scheme secure against adaptive chosen-message attack [GMR88]. Each node also has a unique identifying index. We assume that indices and public keys for all nodes are publicly available in the form of certificates. It is possible to achieve similar security guarantees in a symmetric-key setting with long-term keys.

# 3 VSS for the Hybrid Model—HybridVSS

VSS is the most important part of any distributed key generation environment. Our VSS protocol modifies the AVSS protocol [CKAS02] for our hybrid model. We include recovery messages similar to those from the reliable broadcast protocol by Backes and Cachin [BC03]. We achieve a constant-factor reduction in the protocol complexities using symmetric bivariate polynomials. Further, as described in §1, we use the simpler commitment scheme by Feldman [Fel87] rather than Pedersen's commitment scheme [Ped91].

## 3.1 Protocol Description

Our VSS protocol is composed of a sharing protocol ($\mathsf{Sh}$) and a reconstruction protocol ($\mathsf{Rec}$). In protocol $\mathsf{Sh}$, a dealer $P_d$ upon receiving a $(P_d, \tau, \mathsf{in}, \mathsf{share}, s)$ message, shares a secret $s$, where a counter $\tau$ and the dealer identity $P_d$ forms a unique session identifier. Node $P_i$ finishes the $\mathsf{Sh}$ protocol by outputting a $(P_d, \tau, \mathsf{out}, \mathsf{shared}, C, s_i)$ message, where $C$ is the commitment and $s_i$ is its secret share. Any time after that, upon receiving a message $(P_d, \tau, \mathsf{in}, \mathsf{reconstruct})$, $P_i$ starts the $\mathsf{Rec}$ protocol. The $\mathsf{Rec}$ protocol terminates for a node $P_i$ when $P_i$ outputs a message $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$, where $z_i$ is $P_i$'s reconstructed value of the secret $s$.

**Definition 3.1** *In session $(P_d, \tau)$, protocol VSS in our hybrid model (***HybridVSS***) having an asynchronous network of $n \geq 3t + 2f + 1$ nodes with a $t$-limited Byzantine adversary and $f$-limited crashes and network failures satisfies the following conditions:*

**Liveness:** *If the dealer $P_d$ is honest and finally up in the sharing stage of session $(P_d, \tau)$, then all honest finally up nodes complete protocol $\mathsf{Sh}$.*

**Agreement:** *If some honest node completes protocol $\mathsf{Sh}$ of session $(P_d, \tau)$, then all honest finally up nodes will eventually complete protocol $\mathsf{Sh}$ in session $(P_d, \tau)$. If all honest finally up nodes subsequently start protocol $\mathsf{Rec}$ for session $(P_d, \tau)$, then all honest finally up nodes will finish protocol $\mathsf{Rec}$ in session $(P_d, \tau)$.*

**Consistency:** *Once $t + 1$ honest nodes complete protocol $\mathsf{Sh}$ for session $(P_d, \tau)$, then there exists a fixed value $z$ such that*

- *if the dealer is honest and has shared secret $s$ in session $(P_d, \tau)$, then $z = s$, and*
- *if an honest node $P_i$ reconstructs $z_i$ in session $(P_d, \tau)$, then $z_i = z$.*

**Privacy:** *If an honest dealer has shared secret $s$ in session $(P_d, \tau)$ and no honest node has started the $\mathsf{Rec}$ protocol, then, except with negligible probability, the adversary cannot compute the shared secret $s$.*

**Efficiency:** *The communication complexity for any instance of HybridVSS is d-uniformly bounded.*

*We assume that messages from all the honest and uncrashed nodes are delivered by the adversary.*

Figure 1 describes the $\mathsf{Sh}$ and $\mathsf{Rec}$ protocols for HybridVSS. We use pseudo-code notation and include a special condition **upon** to define actions based on messages received from other nodes or system events. $C$

seem secure against an adaptive attack. This is further supported by the fact that there has been no known adaptive attack for the last twenty years.

```
**Sh protocol for node $P_i$ and session $(P_d, \tau)$**
upon initialization:
  for all $C$ do
    $A_C \leftarrow \emptyset$; $e_C \leftarrow 0$; $r_C \leftarrow 0$
    $c \leftarrow 0$; $c_\ell \leftarrow 0$ for all $\ell \in [1, n]$

upon a message $(P_d, \tau, \mathsf{in}, \mathsf{share}, s)$: /* only $P_d$ */
  choose a random symmetric bivariate polynomial
      $f(x, y) = \sum_{j,\ell=0}^{t} f_{j\ell} x^j y^\ell \in \mathbb{Z}_q[x, y]$
  such that $f_{00} = s$ and $f_{j\ell} = f_{\ell j}$ for $j, \ell \in [0, t]$
  $C \leftarrow C_{j\ell}$ where $C_{j\ell} = g^{f_{j\ell}}$ for $j, \ell \in [0, t]$
  for all $j \in [1, n]$ do
    $a_j(y) \leftarrow f(j, y)$
    send the message $(P_d, \tau, \mathsf{send}, C, a_j)$ to $P_j$

upon a message $(P_d, \tau, \mathsf{send}, C, a)$ from $P_d$ (first time):
  if $\mathsf{verify\text{-}poly}(C, i, a)$ then
    for all $j \in [1, n]$ do
      send the message $(P_d, \tau, \mathsf{echo}, C, a(j))$ to $P_j$

upon a message $(P_d, \tau, \mathsf{echo}, C, \alpha)$ from $P_m$ (first time):
  if $\mathsf{verify\text{-}point}(C, i, m, \alpha)$ then
    $A_C \leftarrow A_C \cup \{(m, \alpha)\}$; $e_C \leftarrow e_C + 1$
    if $e_C = \lceil \frac{n+t+1}{2} \rceil$ and $r_C < t + 1$ then
      Lagrange-interpolate $\overline{a}$ from $A_C$
      for all $j \in [1, n]$ do
        send the message $(P_d, \tau, \mathsf{ready}, C, \overline{a}(j))$ to $P_j$

upon a message $(P_d, \tau, \mathsf{ready}, C, \alpha)$ from $P_m$ (first time):
  if $\mathsf{verify\text{-}point}(C, i, m, \alpha)$ then
    $A_C \leftarrow A_C \cup \{(m, \alpha)\}$; $r_C \leftarrow r_C + 1$
    if $r_C = t + 1$ and $e_C < \lceil \frac{n+t+1}{2} \rceil$ then
      Lagrange-interpolate $\overline{a}$ from $A_C$
      for all $j \in [1, n]$ do
        send the message $(P_d, \tau, \mathsf{ready}, C, \overline{a}(j))$ to $P_j$
    else if $r_C = n - t - f$ then
      $s_i \leftarrow \overline{a}(0)$
      output $(P_d, \tau, \mathsf{out}, \mathsf{shared}, C, s_i)$

upon $(P_d, \tau, \mathsf{in}, \mathsf{recover})$:
  send the message $(P_d, \tau, \mathsf{help})$ to all the nodes
  send all messages in $B$

upon a message $(P_d, \tau, \mathsf{help})$ from $P_\ell$:
  if $c_\ell \leq d(\kappa)$ and $c \leq (t + 1)d(\kappa)$ then
    $c_\ell \leftarrow c_\ell + 1$; $c \leftarrow c + 1$
    send all messages of $B_\ell$
```

```
**Rec protocol for node $P_i$ and session $(P_d, \tau)$**

upon a message $(P_d, \tau, \mathsf{in}, \mathsf{reconstruct})$:
  $c \leftarrow 0$; $S \leftarrow \emptyset$
  for all $j \in [1, n]$ do
    send the message $(P_d, \tau, \mathsf{reconstruct\text{-}share}, s_i)$ to $P_j$

upon a message $(P_d, \tau, \mathsf{reconstruct\text{-}share}, \sigma)$ from $P_m$:
  if $(g^\sigma = \prod_{j=0}^{t} (C_{j0})^{m^j})$ then
    $S \leftarrow S \cup \{(m, \sigma)\}$; $c \leftarrow c + 1$
    if $c = t + 1$ then
      interpolate constant term $(z_i)$ from $S$
      output $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$
      halt
```

Figure 1: Protocol HybridVSS

is a matrix of commitment entries and $e_C$ and $r_C$ are $P_i$'s associated counters for echo and ready messages, respectively. In order to facilitate recovery of the crashed nodes, each node $P_i$ stores all outgoing messages along with their intended recipients in a set $\mathcal{B}$. $\mathcal{B}_\ell$ indicates the subset of $\mathcal{B}$ intended for the node $P_\ell$. Counters $c$ and $c_\ell$ keep track of the numbers of overall help requests and help requests sent by each node $P_\ell$ respectively. We also use the following predicates in our protocol.

**verify-poly**$(C, i, a)$ verifies that the given polynomial $a$ of $P_i$ is consistent with the commitment $C$. Here, $a(y) = \sum_{\ell=0}^{t} a_\ell y^\ell$ is a degree $t$ polynomial. The predicate is true if and only if $g^{a_\ell} = \prod_{j=0}^{t} (C_{j\ell})^{i^j}$ for all $\ell \in [0, t]$.

**verify-point**$(C, i, m, \alpha)$ verifies that the given value $\alpha$ corresponds to the polynomial evaluation $f(m, i)$. It is true if and only if $g^\alpha = \prod_{j,\ell=0}^{t} (C_{j\ell})^{m^j i^\ell}$.

## 3.2 Analysis

The main theorem for HybridVSS is as follows.

**Theorem 3.1** *Assuming the hardness of the discrete-logarithm problem, protocol HybridVSS implements asynchronous verifiable secret sharing in the hybrid model for $n \geq 3t + 2f + 1$.*

We need to show liveness, agreement, consistency, privacy, and efficiency. We combine proof strategies from AVSS [CKAS02, Sec. 3.3] and reliable broadcast [BC03, Sec. 3.3] to achieve this. We start by referring to few lemmas.

**Lemma 3.1 (Lemma 1 [BC03])** *Let $P_i$ be a finally up party during session $(P_d, \tau)$. Then every distinct message sent to $P_i$ by another finally up party $P_j$ during session $(P_d, \tau)$ will be received by $P_i$ in a non-crashed state, provided all associated messages are delivered.*

**Lemma 3.2 (Lemma 2 [CKAS02])** *Suppose an honest node $P_i$ sends a* ready *message containing commitment $C_i$ and a distinct honest node $P_j$ sends a* ready *message containing commitment $C_j$. Then $C_i = C_j$.*

**Liveness** Here, we prove that if the dealer $P_d$ is honest and finally up during the sharing stage of session $(P_d, \tau)$, then all honest finally up nodes complete protocol Sh.

We assume that the dealer $P_d$ is honest and finally up. According to Lemma 3.1, send messages of the form $(P_d, \tau, \mathsf{send}, C, a_i)$ sent by $P_d$ to each finally up node $P_i$ will eventually be received and verified by each such $P_i$. Each of these honest and finally up nodes (at least $n - t - f$) will send an echo message of the form $(P_d, \tau, \mathsf{echo}, C, a_i(j))$ to each system node $P_j$. Using Lemma 3.1, every finally up honest node will thus receive at least $n - t - f$ valid echo messages. A valid echo and ready message is one that satisfies verify-point. As $n - t - f \geq \lceil \frac{n+t+1}{2} \rceil$ for $n \geq 3t + 2f + 1$, every honest finally up node $P_j$ will send ready message $(P_d, \tau, \mathsf{ready}, C, \overline{a}_j(m))$ to every system-node $P_m$ as either the received echo messages are greater than required bound ($\lceil \frac{n+t+1}{2} \rceil$) or it has already received $t+1$ ready messages. As all ready messages will be eventually received by the finally up nodes according to Lemma 3.1, each finally up honest node will receive at least $n - t - f \geq 2t + f + 1$ verifiably correct ready messages. Consequently, each honest finally up node will complete the protocol Sh by outputting $(P_d, \tau, \mathsf{shared})$ messages.

**Agreement** We first show that if some honest node completes protocol Sh of $(P_d, \tau)$, then all honest finally up nodes will eventually complete protocol Sh during session $(P_d, \tau)$. An honest node completes the sharing when it receives $2t + f + 1$ valid ready messages. At least $t + f + 1$ of those have been sent by honest nodes. Using the definitions of verify-poly and verify-point, the honest node sends only valid ready messages. Further, when sending, an honest node sends ready messages to all system nodes. Thus, using Lemma 3.2, every honest finally up node receives at least $t+f+1$ valid ready messages with the same commitment $C$ and sends a ready message containing $C$. Consequently, every honest finally up node receives $n-t-f \geq 2t+f+1$ valid ready messages with commitment $C$ and completes the Sh protocol.

For protocol Rec, we show that if all honest finally up nodes subsequently begin protocol Rec for session $(P_d, \tau)$, then all honest finally up nodes will finish protocol Rec during session $(P_d, \tau)$ by reconstructing $s'_i$. As discussed above, at the end of the sharing step, every honest finally up node $P_i$ computes the same commitment $C$. Moreover, $P_i$ has received enough valid echo or ready messages with commitment $C$ and it computes valid ready messages and a valid share $s_i$ with respect to $C$ ($s_i$ such that ($g^{s_i} = \prod_{j=0}^{t} (C_{j0})^{m^j}$) holds). Thus, if all honest servers subsequently start the reconstruction stage, then every server receives enough valid shares to reconstruct some value, provided the adversary delivers all associated messages.

**Consistency** Once an honest node completes protocol $\mathsf{Sh}$ for session $(P_d, \tau)$, then there exists a fixed value $z$ such that the following holds.

- If the dealer is honest and has shared secret $s$ during session $(P_d, \tau)$, then $z = s$.
- If an honest node $P_i$ reconstructs $z_i$ during session $(P_d, \tau)$, then $z_i = z$.

Suppose an honest dealer has shared a degree-$t$ symmetric bivariate polynomial $f(x, y)$ with constant term equal to the shared secret $s$. As the dealer is honest, an $\mathsf{echo}$ message that an honest node $P_i$ receives from another honest node $P_j$ contains $C, f(j, i)$. As the required number of $\mathsf{echo}$ messages before interpolating the final univariate polynomial at a node is equal to $\lceil \frac{n+t+1}{2} \rceil$, it is impossible for faulty nodes to make a node accept commitment $C'$ different from commitment $C$ suggested by the dealer. Subsequently, such an honest node $P_i$, after verification with $\mathsf{verify\text{-}point}$, interpolates a polynomial $\overline{a}(y)$ such that $\overline{a}(y) = f(i, y)$. Assume an honest node receives $t + 1$ $\mathsf{ready}$ messages before obtaining $\lceil \frac{n+t+1}{2} \rceil$ commitment $C$ $\mathsf{echo}$ messages. Using Lemma 3.2 all these $\mathsf{ready}$ messages have same commitment and with at least of one of them from an honest node, it is equal to $C$. The honest node will interpolate the same $\overline{a}(y)$ as in the case of the $\mathsf{echo}$ messages. Using the agreement property, if a node completes the protocol $\mathsf{Sh}$, then all honest nodes will eventually finish it. Let $\mathcal{S}$ be any set of $t + 1$ honest nodes $(P_j)$ that have finished the sharing. Let $s_{j,d}$ represent the share for node $P_j$ such that $s_{j,d} = \overline{a}(0) = f(j, 0)$. Let $\lambda_j^{\mathcal{S}}$ be Lagrange interpolation coefficients for the set $\mathcal{S}$ and position 0. We have

$$\begin{aligned} z &= \sum_{P_j \in \mathcal{S}} \lambda_j^{\mathcal{S},0} s_{j,d} \\ &= \sum_{P_j \in \mathcal{S}} \lambda_j^{\mathcal{S},0} f(j, 0) \\ &= s \end{aligned}$$

and if the dealer is honest and has shared secret $s$ during session $(P_d, \tau)$, then $z = s$.

To prove the second part, assume that two distinct honest servers $P_i$ and $P_j$ reconstruct values $z_i$ and $z_j$ such that $z_i \neq z_j$. This means that they are interpolated from two distinct sets $S_i = \{\ell, s_\ell^{(i)}\}$ and $S_j = \{\ell, s_\ell^{(j)}\}$ of $t + 1$ shares each, which are valid with respect to the unique commitment $C$ (using Lemma 3.2) used by $P_i$ and $P_j$. As the shares in $S_i$ and $S_j$ are verified against commitment $C$ and they are valid, it is easy to see that $g^{z_i} = C_{00} = g^{z_j}$. As $g$ is a generator for a prime order group, $z_i = z_j$ and the assumption that $z_i \neq z_j$ is wrong.

**Privacy** To prove the privacy property, we use the discrete logarithm assumption. The adversary's view consists of polynomials $f(i, y)$ for these Byzantine nodes $i$, and the commitment matrix $C$ and the generator $g$ provided by the dealer. Assume that there is an adversary algorithm $\mathcal{A}$ that can compute the shared secret $s$ given $g$, $C$ and $t$ degree-$t$ polynomials consistent with $C$. We prove that a challenger $\mathcal{B}$ with an oracle access to such an adversary algorithm $\mathcal{A}$ can solve any instance $(p, q, g, g^x)$ of discrete logarithm problem.

Given a discrete logarithm instance $(p, q, g, g^x)$, the challenger $\mathcal{B}$ generates $t$ degree-$t$ polynomials $r_i(y) \in Z_q[y]$ and associates them with non-zero indices $i$. It then computes $g^{r_i(0)}$ for each index $i$, sets $g^{r_0(0)} = g^x$ and computes $C_{0,k} = C_{k,0}$ for $k = [0, t]$ using the inverse-interpolation method.[3] Proceeding similarly, for each $0 < \ell \leq t$, it uses $C_{0,\ell}$ and $g^{r_i(\ell)}$ for each index $i$ and computes $C_{\ell,k} = C_{k,\ell}$ and completes the symmetric commitment matrix $C$ which is consistent with $g^x$ as $C_{0,0}$ and polynomials $r_i(y)$. $\mathcal{B}$ can then present this matrix $C$ along with polynomial $r_i$ to the adversary algorithm $\mathcal{A}$ and return the output $s = x$ as the discrete logarithm value for tuple $(p, q, g, g^x)$. As this is not possible, except with negligible probability, we prove that if an honest dealer has shared secret $s$ during session $(P_d, \tau)$ and no honest node has started the $\mathsf{Rec}$ protocol, then the adversary cannot compute the secret $s$ except with negligible probability.

[3] This involves computing polynomial coefficients in an exponentiated form given evaluations of that degree-$t$ polynomial at $t + 1$ points in an exponentiated form.

**Efficiency Discussion** Initially, we discuss complexities when there are no crashes. A protocol execution without any crashes has $O(n^2)$ message complexity and $O(\kappa n^4)$ communication complexity where the size of the message is dominated by commitment matrix $C$ having $t(t+1)/2 = O(n^2)$ entries. Using a collision-resistant hash function, Cachin el al. [CKAS02, Sec. 3.4] suggest a way to reduce the communication complexity to $O(\kappa n^3)$. In this approach, commitments are generated using the exponentiated form of bivariate polynomial evaluations ($A_j^{(i)} = g^{f(i,j)}$). Let $A^{(j)} = \langle A_0^{(j)}, A_1^{(j)}, \ldots, A_n^{(j)} \rangle$. In this case, the communication complexity gets reduced by a linear factor using the $A^{(0)}$ vector and a vector $h = \langle h_0, h_1, \ldots, h_n \rangle$, where $H$ is collision-resistant hash function and $h_j = H(A^{(j)})$. We use this modification in our implementation and refer readers to [CKAS02, Sec. 3.4] for a detailed description.[4]

Now, assume there are crashes and there are subsequent recoveries. As defined earlier, $d(\kappa)$ bounds the number of possible crashes in the system. In addition, each of the Byzantine nodes may produce unlimited false $\mathsf{help}$ messages, out of which first $d(\kappa)$ will be answered by honest nodes. Therefore, each honest node will in total answer up to $(t+1)d(\kappa)$ $\mathsf{help}$ messages. The recovery mechanism requires $O(n^2)$ messages from the recovering node and $O(n)$ messages from each helper node. Therefore, the total message and communication complexity of HybridVSS are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively and we obtain a uniform polynomial bound on the communication complexity.

# 4 Distributed Key Generation—DKG

HybridVSS requires a dealer ($P_d$) to select a secret and to initiate a sharing. DKG, going one step further, generates a secret in a completely distributed fashion, such that none of the system nodes knows the secret, while any $t+1$ nodes can combine their shares to determine it. Although it seems that a DKG is just a system with $n$ nodes running their VSSs in parallel and summing all the received shares together at the end, it is not that simple in an asynchronous setting. Agreeing on $t+1$ or more VSS instances such that all of them will finish for all the honest nodes (the *agreement on a set* problem [BOCG93]), and the difficulty of differentiating between a slow node and a faulty node in the asynchronous setting are the primary sources of the added complexity.

In our hybrid system model, with no timing assumption, the node cannot wait for more than $n-t-f$ VSSs to finish. The adversary can certainly make agreeing on a subset of size $t+1$ among those nodes impossible, by appropriately delaying its messages. Cachin et al. [CKAS02] solves a similar agreement problem in their proactive refresh protocol using a multi-valued validated Byzantine agreement (MVBA) protocol. Known MVBA protocols [CKPS01] require threshold signature and threshold coin-tossing primitives [CKS00] and the suggested algorithms for both of these primitives require either a dealer or a DKG. As we aim to avoid the former in this paper and the latter is our aim itself, we cannot use their MVBA protocol. Randomization in the form of distributed coin tossing or equivalent randomization functionality is necessary for an expected constant-round Byzantine agreement; it thwarts the attack possible with an adversary knowing the predefined node selection order. However, an efficient algorithm for dealerless distributed coin tossing without a DKG is difficult to achieve[5], and we refrain from using randomized Byzantine agreement protocols.

We follow a much simpler approach with the same communication complexity as MVBA protocols. We use a leader-initiated reliable broadcast system with a faulty-leader change facility, inspired by Castro and Liskov's view-change protocol [CL02]. We choose this (optimistic phase + pessimistic phase) approach, as we expect the Byzantine failures to be infrequent in practice. The probability that the current leader of the system is not behaving correctly is small and it is not worth spending more time and bandwidth by broadcasting proposals by all the nodes during every MVBA. With this background, we now define and analyze our DKG protocol.

## 4.1 Protocol Description.

In our DKG protocol, for session $\tau$ and leader $\mathcal{L}$, each node $P_d$ selects a secret value $s_d$ and shares it among the group using protocol $\mathsf{Sh}$ of HybridVSS for session $(P_d, \tau)$. Each node finishes the DKG protocol

[4] As we do not use Pedersen's commitment, we do not need $B^{(j)}$ and $h_b$ vectors, here.

[5] Canetti and Rabin [CR93] define a dealerless distributed coin tossing without a DKG; however, their protocol requires $n^2$ VSSs for each coin toss and is consequently inefficient.

by outputting a $(\overline{\mathcal{L}}, \tau, \mathsf{DKG\text{-}completed}, C, s_i)$ message, where $s_i$ and $C$ are its share and the commitment respectively and $\overline{\mathcal{L}} = \mathcal{L}$ or a subsequent leader

**Definition 4.1** *In session $\tau$, protocol* **DKG** *in our hybrid model having an asynchronous network of $n \geq 3t + 2f + 1$ nodes with a $t$-limited Byzantine adversary and $f$-limited crashes and network failures satisfies the following conditions:*

**Liveness:** *All honest finally up nodes complete protocol DKG in session $\tau$, except with negligible probability.*

**Agreement:** *If some honest node completes protocol DKG in session $\tau$, then, except with negligible probability, all honest finally up nodes will eventually complete protocol DKG in session $\tau$.*

**Consistency:** *Once an honest node completes the DKG protocol for session $\tau$, then there exists a fixed value $s$ such that, if an honest node $P_i$ reconstructs $z_i$ in session $\tau$, then $z_i = s$.*

**Privacy:** *If no honest node has started the* Rec *protocol, then, except with negligible probability, the adversary cannot compute the shared secret $s$.*

**Efficiency:** *The communication complexity for any instance of DKG is $d$-uniformly bounded.*

*We assume that messages from all the honest and uncrashed nodes are delivered by the adversary.*

We first describe the optimistic phase of our DKG protocol. For each session $\tau$, one among $n$ nodes works as a leader. The leader $\mathcal{L}$, once it finishes the VSS proposal by $t+1$ nodes with $(P_d, \tau, \mathsf{out}, \mathsf{shared}, C_d, s_{i,d})$, broadcasts the $n - t - f$ ready messages (set $\widehat{\mathcal{R}}$) it received for each of those $t+1$ finished VSSs (set $\widehat{\mathcal{Q}}$). Nodes include signatures with ready messages to enable the leader to provide a validity proof for its proposal. In this *extended HybridVSS* protocol, shared messages look like $(P_d, \tau, \mathsf{out}, \mathsf{shared}, C_d, s_{i,d}, \mathcal{R}_d)$, where a set $\mathcal{R}_d$ includes $n - t - f$ signed ready messages for session $(P_d, \tau)$. Once this broadcast completes, each node knows $t+1$ VSS instances to wait for. Once a node $P_i$ finishes those, it sums the shares $s_{i,d}$ to obtain its final share $s_i$.

If the leader is faulty and does not proceed with the protocol or sends arbitrary messages, the protocol enters into a pessimistic phase. Here, other nodes use a *leader-change* mechanism to change their leader with a pre-defined cyclic permutation ($\pi$) and provide liveness without damaging system safety. Every unsatisfied node sends a signed leader-change (lead-ch) request to all the nodes for the next leader $\pi(\mathcal{L})$ if it receives an invalid message from the existing leader $\mathcal{L}$ or if its timer timed out. Timeouts are based on the function $delay(t)$ described in §2.1. When a node collects $t+1$ lead-ch messages for leaders $> \mathcal{L}$, it is confirmed that at least one honest node is unsatisfied and it sends a lead-ch message to all the nodes for the smallest leader among those requested, if it has not done that yet. Once a node receives $n-t-f$ lead-ch requests for a leader $\overline{\mathcal{L}} > \mathcal{L}$, it accepts $\overline{\mathcal{L}}$ as the new leader and enters into the optimistic phase. The new leader also enters into the optimistic phase and sends a send message for set $\overline{\mathcal{Q}}$ if it is non-empty or else for set $\widehat{\mathcal{Q}}$. Set $\overline{\mathcal{M}}$ contains $\lceil \frac{n+t+1}{2} \rceil$ signed echo messages or $t+1$ signed ready messages for the associated set $\overline{\mathcal{Q}}$ of completed VSSs. Set $\overline{\mathcal{Q}}$ avoids two honest nodes finishing with two different VSSs sets, and set $\overline{\mathcal{M}}$ avoids false $\overline{\mathcal{Q}}$ sets from the dishonest nodes. While sending its proposal, $\mathcal{L}$ also includes lead-ch signatures received from $n-t-f$ nodes to prove its validity to the nodes who have not received enough lead-ch messages. As in HybridVSS, the set $\mathcal{B}$ contains all outgoing messages at a node along with their intended recipients and $\mathcal{B}_\ell$ represents the subset of messages destined for node $P_\ell$. Counters $c$ and $c_\ell$ keep track of the numbers of overall help requests and help requests sent by each node $P_\ell$ respectively. Figure 2 presents the optimistic and the pessimistic phases of the DKG protocol. Protocol Rec remains exactly the same.

## 4.2 Analysis.

The main theorem for our DKG is as follows.

**Theorem 4.1** *Assuming the hardness of the discrete-logarithm problem, protocol DKG provides an asynchronous distributed key generation mechanism in the hybrid model for $n \geq 3t + 2f + 1$.*

We need to show the liveness, agreement, consistency, privacy, and efficiency of our DKG.

**Optimistic phase for node $P_i$ in Session ($\tau$) with Leader $\mathcal{L}$**

```
upon initialization:
  $e_{\mathcal{Q}} \leftarrow 0$; $r_{\mathcal{Q}} \leftarrow 0$ for every $\mathcal{Q}$
  $\overline{\mathcal{Q}} \leftarrow \emptyset$; $\widehat{\mathcal{Q}} \leftarrow \emptyset$
  $\overline{\mathcal{M}} \leftarrow \widehat{\mathcal{R}} \leftarrow n-t-f$ signed lead-ch messages for $\mathcal{L}$
  $c \leftarrow 0$; $c_\ell \leftarrow 0$ for all $\ell \in [1,n]$
  $lc_{\mathcal{L}} \leftarrow 0$ for each $\mathcal{L}$; $lc_{flag} \leftarrow$ false; $\mathcal{L}_{++} \leftarrow \pi^{-1}(\mathcal{L})$
  for all $d \in [1,n]$ do
    initialize extended-HybridVSS Sh protocol $(P_d, \tau)$

upon $(P_d, \tau, \mathsf{out}, \mathsf{shared}, C_d, s_{i,d}, \mathcal{R}_d)$ (first time):
  $\widehat{\mathcal{Q}} \leftarrow \{P_d\}$; $\widehat{\mathcal{R}} \leftarrow \{\mathcal{R}_d\}$
  if $|\widehat{\mathcal{Q}}| = t+1$ and $\overline{\mathcal{Q}} = \emptyset$ then
    if $P_i = \mathcal{L}$ then
      send the message $(\mathcal{L}, \tau, \mathsf{send}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to each $P_j$
    else
      $delay \leftarrow delay(t)$; start timer($delay$)

upon a message $(\mathcal{L}, \tau, \mathsf{send}, \mathcal{Q}, \mathcal{R}/\mathcal{M})$ from $\mathcal{L}$ (first time):
  if verify-signature($\mathcal{Q}, \mathcal{R}/\mathcal{M}$) then
    if $\overline{\mathcal{Q}} = \emptyset$ or $\overline{\mathcal{Q}} = \mathcal{Q}$ then
      send the message $(\mathcal{L}, \tau, \mathsf{echo}, \mathcal{Q})_{\mathsf{sign}}$ to each $P_j$

upon a message $(\mathcal{L}, \tau, \mathsf{echo}, \mathcal{Q})_{\mathsf{sign}}$ from $P_m$ (first time):
  $e_{\mathcal{Q}} \leftarrow e_{\mathcal{Q}} + 1$
  if $e_{\mathcal{Q}} = \lceil \frac{n+t+1}{2} \rceil$ and $r_{\mathcal{Q}} < t+1$ then
    $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow \lceil \frac{n+t+1}{2} \rceil$ signed echo messages for $\mathcal{Q}$
    send the message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ to each $P_j$

upon a message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ from $P_m$(first time):
  $r_{\mathcal{Q}} \leftarrow r_{\mathcal{Q}} + 1$
  if $r_{\mathcal{Q}} = t+1$ and $e_{\mathcal{Q}} < \lceil \frac{n+t+1}{2} \rceil$ then
    $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow t+1$ signed ready messages for $\mathcal{Q}$
    send the message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ to each $P_j$
  else if $r_{\mathcal{Q}} = n-t-f$ then
    stop timer, if any
    wait for shared output-messages for each $P_d \in \mathcal{Q}$
    $s_i \leftarrow \sum_{P_d \in \mathcal{Q}} s_{i,d}$; $\forall_{p,q} : C_{p,q} \leftarrow \prod_{P_d \in \mathcal{Q}} (C_d)_{p,q}$
    output $(\mathcal{L}, \tau, \mathsf{DKG\text{-}completed}, C, s_i)$

upon timeout
  if $lc_{flag}$ = false then
    if $\overline{\mathcal{Q}} = \emptyset$ then
      send msg $(\tau, \mathsf{lead\text{-}ch}, \pi(\mathcal{L}), \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})_{\mathsf{sign}}$ to each $P_j$
    else
      send msg $(\tau, \mathsf{lead\text{-}ch}, \pi(\mathcal{L}), \overline{\mathcal{Q}}, \overline{\mathcal{M}})_{\mathsf{sign}}$ to each $P_j$
    $lc_{flag} \leftarrow$ true

upon $(\mathcal{L}, \tau, \mathsf{in}, \mathsf{recover})$:
  send the message $(\mathcal{L}, \tau, \mathsf{help})$ to all the nodes.
  send all messages in $B_{\mathcal{L},\tau}$
upon a message $(\mathcal{L}, \tau, \mathsf{help})$ from $P_\ell$:
  if $c_\ell \leq d(\kappa)$ and $c \leq (t+1)d(\kappa)$ then
    $c_\ell \leftarrow c_\ell + 1$; $c \leftarrow c + 1$
    send all messages of $B_{\ell(\mathcal{L},\tau)}$
```

**Leader-change for node $P_i$ in session ($\tau$) with Leader $\mathcal{L}$**

```
upon a msg $(\tau, \mathsf{lead\text{-}ch}, \overline{\mathcal{L}}, \mathcal{Q}, \mathcal{R}/\mathcal{M})_{\mathsf{sign}}$ from $P_j$(first time):
  if $\overline{\mathcal{L}} > \mathcal{L}$ and verify-signature($\mathcal{Q}, \mathcal{R}/\mathcal{M}$) then
    $lc_{\overline{\mathcal{L}}} \leftarrow lc_{\overline{\mathcal{L}}} + 1$; $\mathcal{L}_{++} \leftarrow \min(\mathcal{L}_{++}, \overline{\mathcal{L}})$
    if $\mathcal{R}/\mathcal{M} = \mathcal{R}$ then $\widehat{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\widehat{\mathcal{R}} \leftarrow \mathcal{R}$
    else $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow \mathcal{M}$
    if ($\sum lc_{\mathcal{L}} = t+1$ and $lc_{flag}$ = false) then
      if $\overline{\mathcal{Q}} = \emptyset$ then
        send the msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{++}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to each $P_j$
      else
        send the msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{++}, \overline{\mathcal{Q}}, \overline{\mathcal{M}})$ to each $P_j$
    else if ($lc_{\overline{\mathcal{L}}} = n-t-f$) then
      $\overline{\mathcal{M}} \leftarrow \widehat{\mathcal{R}} \leftarrow n-t-f$ signed lead-ch messages for $\overline{\mathcal{L}}$
      $\mathcal{L} \leftarrow \overline{\mathcal{L}}$; $lc_{\mathcal{L}} \leftarrow 0$; $\mathcal{L}_{++} \leftarrow \pi^{-1}(\mathcal{L})$; $lc_{flag}$ = false
      if $P_i = \mathcal{L}$ then
        if $\overline{\mathcal{Q}} = \emptyset$ then
          send the message $(\mathcal{L}, \tau, \mathsf{send}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to each $P_j$
        else
          send the message $(\mathcal{L}, \tau, \mathsf{send}, \overline{\mathcal{Q}}, \overline{\mathcal{M}})$ to each $P_j$
      else
        $delay \leftarrow delay(t)$; start timer($delay$)
```

Figure 2: DKG Protocol

**Liveness.** In HybridVSS, if the dealer is honest and finally up, then all honest finally up nodes complete the sharing initiated by it. With $n - t - f$ honest finally up nodes in the system, each honest finally up node will eventually complete $t+1$ HybridVSS sharings, as required. Each such node will start a timer upon completing these $t+1$ HybridVSSs. If the leader is honest and uncrashed, it also completes $t+1$ HybridVSSs, and broadcasts its proposal; based on the liveness property of reliable broadcast [BC03], each honest finally up node delivers the same verifiable proposal. Honest finally up nodes stop their timers when they complete this reliable broadcast. To finish, according to the HybridVSS agreement properties, all honest finally up nodes complete protocol $\mathsf{Sh}$ for nodes in this proposal.

If the leader is compromised, crashed or does not finish its broadcast before a timeout at an honest node, then a signed $\mathsf{lead\text{-}ch}$ request is broadcasted by that honest node (pessimistic phase). After receiving $n - t - f$ $\mathsf{lead\text{-}ch}$ requests, the new leader takes over, each honest node starts a timer for the proposal from the new leader, and the protocol reenters the optimistic phase. As the number of crashes is polynomially bounded and the network eventually gets repaired resulting in message delays becoming eventually bounded by $delay(t)$, an honest finally up leader will eventually reliably broadcast a proposal and protocol DKG will complete. The requirement of $n - t - f$ $\mathsf{lead\text{-}ch}$ requests for a leader replacement makes sure that nodes do not complete the leader-change too soon. An honest node sends a signed $\mathsf{lead\text{-}ch}$ message for the smallest leader (among the received set) if it receives $t + 1$ $\mathsf{lead\text{-}ch}$ messages, even if it has not observed any fault, as this indicates that at least one honest node has observed some fault and the node does not want to start the leader-change too late.

**Agreement.** An honest node completes DKG when it completes a reliable broadcast of the current leader's sharing proposal, finishes HybridVSS sharing by $t+1$ nodes in that proposal, and computes its final share as a summation of the shares obtained from these $t + 1$ sharings. According to the agreement property of the reliable broadcast, if one honest node completes the protocol, then all honest finally up nodes will eventually complete the protocol. Further, when only some (but not all) nodes complete the reliable broadcast before a leader-change, sets $\overline{\mathcal{Q}}$ and $\overline{\mathcal{M}}$ ensure that all nodes complete a reliable broadcast for same $\overline{\mathcal{Q}}$ after the leader-change. According to the agreement property of the HybridVSS, once an honest node completes a set of $t + 1$ sharings, then all honest finally up nodes will eventually complete all of these $t + 1$ sharings. Consequently, once an honest node completes DKG, then all honest finally up nodes will eventually complete the DKG protocol.

**Consistency.** According to the agreement property, once an honest node completes the DKG for session $(\tau)$, then all $(n - t - f)$ honest finally up nodes will eventually complete the DKG protocol. According to the consistency property of the reliable broadcast protocol, each of these nodes will decide the same set of $t + 1$ sharings. Further, when only some (but not all) nodes complete the reliable broadcast before a leader-change, sets $\overline{\mathcal{Q}}$ and $\overline{\mathcal{M}}$ make sure that all nodes complete a reliable broadcast for same $\overline{\mathcal{Q}}$ after the leader-change. For each of the completed sharings, if run individually, each node $P_i$ will reconstruct the same shared secret $z_{i,d}$ where $P_d$ is the dealer for the sharing. Let $z_i = \sum_{P_d \in \mathcal{Q}} z_{i,d}$. As nodes add their shares for the completed $t + 1$ sharings as the DKG finishes and as Lagrange-interpolation is homomorphic over addition, on reconstruction after the DKG protocol each node will output the same $z_i = z$(say).

**Privacy.** In DKG, sharings by $t+1$ nodes are used, where at least one of those shared secrets was proposed by an honest party. In a reliable broadcast, two honest nodes always finish protocol with the same message; therefore, the same $t + 1$ sharings are completed by all the honest nodes. For a HybridVSS execution, if the dealer $P_d$ is honest then until the reconstruction protocol starts, the adversary cannot compute the shared secret $s_d$. Therefore, at the end of DKG protocol, the adversary does not know at least one of the $t + 1$ shared secrets. As the system's secret $s = \sum_{P_d \in \mathcal{Q}} s_d$, the adversary cannot compute the shared secret $s$.

**Efficiency.** The message and communication complexities of HybridVSS are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively. In the DKG protocol with the asynchronous communication assumption, the system may complete all $n$ VSS executions, even though the required execution count is just $t + 1$; thus, the message and communication complexities of the $n$ HybridVSS $\mathsf{Sh}$ protocols in DKG are $O(tdn^3)$ and $O(\kappa tdn^4)$ respectively. If the DKG protocol completes without entering into the pessimistic phase, then the system only needs an additional reliable broadcast of message of size $O(\kappa n)$, message complexity $O(tdn^2)$ and communication complexity $O(\kappa tdn^3)$. As a result, the optimal message and communication complexities for the DKG protocol are $O(tdn^3)$ and $O(\kappa tdn^4)$ respectively. In the pessimistic case, the total number of leader changes is bounded by $O(d)$. Each leader change involves $O(tdn^2)$ messages and $O(\kappa tdn^3)$ communication bits. For each faulty leader, $O(tdn^2)$ messages and $O(\kappa tdn^3)$ bits are communicated during its administration. Therefore, in the worst case, $O(td^2n^2)$ messages and $O(\kappa td^2n^3)$ bits are communicated before the DKG completes and worst case message and communication complexities of the DKG protocol are $O(tdn^2(n+d))$ and $O(\kappa tdn^3(n+d))$ respectively. Note that considering just a $t$-limited Byzantine adversary (and not also crashes and link failures), the above complexities become $O(n^3)$ and $O(\kappa n^4)$ respectively. These are same as the complexities of the proactive refresh protocol for AVSS [CKAS02].

# 5 Realizing Proactiveness

In proactive security, nodes modify their shares at phase changes such that an adversary's knowledge of $t$ shares from one phase becomes useless in the next phase. Here, although the adversary is restricted to $t$ nodes during any phase, it may corrupt more than $t$ nodes in its complete lifetime without learning anything about the secret. In this section, to realize proactiveness in our DKG system, we introduce the notion of phase in our hybrid model (§2) and design share renewal and recovery protocols.

## 5.1 System Model

**Common Phase**

In the asynchronous communication model, without a common clock, realizing the concept of a common phase is difficult. Similar to Cachin et al. [CKAS02], we use *local clocks* with clock ticks at pre-defined intervals. The number of clock ticks received by a honest node defines its local phase. In order to achieve the required synchronization without hampering safety, nodes start the proactive protocol with their local clock tick, but wait for $t$ other nodes to start the phase before proceeding with it.

Due to the eventual nature of the liveness condition, any timing constraint always affects liveness of an asynchronous protocol. A share renewal protocol in our model might not terminate within the same phase. It is possible to achieve liveness at the cost of safety/privacy by continuing with the shares from the previous phase until new shares are determined. However, we give importance to safety rather than liveness and system nodes delete their shares as the renewal protocol starts; there is no phase overlap.

**Byzantine Adversary**

The adversary can corrupt at most $t$ nodes in any local phase $\tau \geq 0$. We assume that it is possible to remove the adversary from a node by rebooting it in a trusted way using a read-only device. As the adversary could have extracted the private key from a recovering node, once rebooted the node should ask the CA to put its old certificate on its certificate revocation list, generate a new key pair and get the new public key signed.

To maintain liveness in a proactive system with simultaneous Byzantine and crash-recovery nodes, we assume that the crash-recovery time is more than the message transfer delay between two uncrashed nodes; specifically, the time the adversary takes to shift from one crashed node to another is larger than required by a send message between two honest uncrashed nodes. Note that this assumption is required exclusively due to crash-recovery and link failure assumption; we justify it in §5.2. The adversary may continue to hold a node in consecutive phases.

It is also possible to use an asynchronous proactive secure message transmission mechanism [BCS03] to avoid frequent public-private key pair modifications. However, this requires a hardware secure co-processor.

**Forward Secrecy**

If a private communication channel between two honest nodes is not forward secret, the adversary may decipher their secret communication by compromising one of them in a later phase. To overcome this problem, we use an ephemeral Diffie-Hellman cipher suite while creating TLS links and reconstruct them at the start of each new phase. This makes sure that a message sent in a local phase $\tau$ of the sender is delivered to the receiver in the same local phase or it is lost.

## 5.2 Share Renewal Protocol

A share renewal protocol enables DKG nodes to renew their shares such that protocol $\mathsf{Rec}$ will output the same secret and the adversary does not learn anything about it.

**Protocol Description.**

**Definition 5.1** *Suppose system nodes hold shares of a secret $s$ shared using a VSS protocol (§3) for a phase $\tau - 1$. An asynchronous* share renewal *protocol (Renew) for phase $\tau$ in a hybrid model having a network of $n \geq 3t + 2f + 1$ nodes with $t$-limited Byzantine adversary and $f$-limited crashes and network failures satisfies the following conditions:*

**Liveness:** *If the adversary delivers all associated messages within phase $\tau$, then all honest nodes complete the Renew protocol, except with negligible probability.*

**Consistency:** *If at least $t + 1$ honest nodes complete the Renew protocol during phase $\tau$ before detecting a subsequent clock tick, the system maintains a verifiable sharing of $s$.*

**Privacy:** *The $t$-limited adversary cannot compute the shared secret $s$ after an execution of the Renew protocol, except with negligible probability.*

**Efficiency:** *The communication complexity for any instance of verifiable share renewal is $d$-uniformly bounded.*

As already discussed, if the adversary restricts the network from delivering all the protocol messages within the phase, privacy is still preserved, but the secret may get lost, hampering the liveness. This definition is analogous to the definition of an asynchronous secure refresh protocol by Cachin et al.[CKAS02] We design a share renewal protocol by making three modifications to our DKG, which are motivated by the refresh protocol in [CKAS02].

- On receiving a clock tick for phase $\tau$, instead of running the HybridVSS protocol for a random key, node $P_i$ reshares its share $s_{i,\tau-1}$ from phase $\tau - 1$. It then erases the old share, the bivariate polynomial used during resharing, and the univariate polynomials from the $\mathsf{send}$ messages, and broadcasts its clock tick. While retransmitting $\mathsf{send}$ messages during a node recovery, only the commitments are sent.
- A node waits for $t + 1$ identical clock ticks before proceeding with protocol $\mathsf{Sh}$ instances.
- Once a node $P_i$ receives $n - t - f$ $\mathsf{ready}$ messages for a decided set $\mathcal{Q}$, instead of adding shares $s_{i,d}$ for $P_d \in \mathcal{Q}$, it Lagrange-interpolates them for index 0 to obtain the new share. Commitments are accordingly modified as $V_\ell = \prod_{P_d \in \mathcal{Q}} ((C_{d,\tau})_{\ell 0})^{\lambda_d^{\mathcal{Q},0}}$ for $\ell \in [0, t]$.

The main theorem for protocol Renew is as follows.

**Theorem 5.1** *Assuming the hardness of the discrete-logarithm problem, the Renew protocol implements asynchronous verifiable share renewal in the hybrid model for $n \geq 3t + 2f + 1$.*

We need to show liveness, consistency, privacy, and efficiency.

**Liveness.** Expect for a small modification to maintain forward secrecy [DvOW92], the liveness analysis is exactly same as that for the DKG protocol. We delete the univariate polynomials from the $\mathsf{send}$ messages stored to facilitate recovery, as their compromise can lead to compromise of the node's previous-phase share and subsequently the system's secret. However, this does not affect the liveness of the system. We observe that among the $\lceil (n + t + 1)/2 \rceil$ $\mathsf{echo}$ messages received during an $\mathsf{Sh}$ instance of HybridVSS, each honest node need only receive $t + 1$ consistent shares of its univariate polynomial in order that the protocol $\mathsf{Sh}$ can continue. With the assumption that at least $t + 1$ honest and uncrashed nodes receive the $\mathsf{send}$ messages transmitted by an honest and uncrashed node before the adversary can crash them, liveness is guaranteed for each such $\mathsf{Sh}$ instance. The remaining liveness discussion remains exactly the same as that of the DKG protocol.

**Consistency.** According to the consistency property of protocol DKG, for each of the completed sharings, if run individually, each node $P_i$ will reconstruct same shared secret $z_{i,d}$ where $P_d \in \mathcal{Q}$ is the dealer for the sharing. Here, instead of summing shares received from all $t+1$ selected dealers, nodes Lagrange-interpolate them for index 0. As Lagrange-interpolation is homomorphic to addition and scalar multiplication, as with DKG, during reconstruction each node will reconstruct same secret (say) $z$.

We also need to prove that $z = s$. Let $\mathcal{S}$ represent a set of $t+1$ nodes that completes the Renew protocol in a phase $\tau$ and have not yet received the next clock tick. Here,

$$\begin{aligned}
z &= \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} s_{i,\tau} \\
&= \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} \Big( \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},0} s_{i,d} \Big) \\
&= \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},0} \Big( \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} s_{i,d} \Big) \\
&= \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},0} s_{d,\tau-1} \\
&= s
\end{aligned}$$

Thus, if at least $t+1$ honest nodes complete the share renewal protocol during phase $\tau$ before detecting a subsequent clock tick, the system maintains a verifiable sharing of $s$.

**Privacy.** Privacy is proven in exactly the same way as the privacy of our DKG protocol.

**Efficiency.** The efficiency discussion remains similar to the efficiency discussion in the DKG protocol. The worst-case message and communication complexities of the Renew protocol are $O(tdn^2(n+d))$ and $O(\kappa tdn^3(n+d))$ respectively.

### 5.3 Share Recovery Protocol

The adversary may crash, isolate or compromise some of the nodes. This may get detected by the node itself or by the system as a whole using the techniques beyond our scope. After detection of crash and compromise, a node will be rebooted using read-only memory, which however does not provide it with its share. In a proactive DKG system, the ability of a node to recover its lost share, when rebooted as above or alienated from the part of the network, must be ensured. Otherwise, the adversary can destroy the complete system by gradually crashing or isolating $n-t$ nodes.

The recover and help messages in our HybridVSS, DKG and share renewal protocols suffice to handle share recoveries. To achieve automatic share recovery upon reboot, we add a recover message to nodes' reboot procedure.

## 6 Group Modification Protocols

On a long-term basis, it is inevitable that the set of nodes in the system will need to be modified; new nodes may join or old nodes may leave. To maintain the resilience bound $n \geq 3t+2f+1$, this may also lead to a modification in the security threshold $t$ or the crash-limit $f$ of the system. Here, we present protocols to achieve node addition, node removal, and security-threshold and crash-limit modification.

### 6.1 Group Modification Agreement

For group modification protocols, it is important to include a mechanism to propose and agree on group modification proposals. Leaving this to node administrators can not only create bottlenecks in the system, but it can also provide new avenues of attack. Using reliable broadcast methodology, we propose a simple agreement protocol for this. To avoid inefficient atomic or causal broadcast primitives [HT93], we impart commutativity to the group modification proposals. Node addition and removal operations are commutative in nature; however, the threshold and crash-limit modifications are not. We solve this problem by attaching threshold and crash-limit modification requests to node addition or removal proposals. With every node addition or removal proposal, a proposer has to specify whether change in the size of the group made by its proposal should affect the security-threshold or the crash-limit. An interested node will send such a proposal to all the nodes and nodes who agree with the proposal continue with echo messages from a reliable broadcast [BC03]. Once it receives $n - t - f$ ready messages, a node adds the proposal into its modification queue. Like other proactive protocols, assuming that the $n-t-f$ nodes finish with the same set of proposals during a phase, liveness is assured; additionally, safety is always assured.

## 6.2 Node Addition

We can increase the redundancy of the system by adding new nodes. It is easily possible to provide shares to the added nodes at the start of a new phase by including those into the list of nodes. Although the new nodes cannot contribute with send messages, for any node-additions with new threshold smaller than the old honest-uncrashed count, sufficient renewal proposals are available.

However, considering possible large durations of phases or even the absence of proactivity, we need a node-addition protocol that does not rely on share renewal.

**Protocol Description.**

**Definition 6.1** *Suppose system nodes hold shares of a secret $s$ shared using the DKG protocol (§3) for a phase $\tau$. A* node addition *protocol (NodeAdd) to add a new honest node $P_{new}$ during phase $\tau$ in a hybrid model having a network of $n \geq 3t + 2f + 1$ nodes with $t$-limited Byzantine adversary and $f$-limited crashes and network failures satisfies the following conditions:*

**Liveness:** *If the adversary delivers all associated messages within phase $\tau$, then the new honest node $P_{new}$ completes the protocol NodeAdd, except with negligible probability.*

**Consistency:** *Once node $P_{new}$ completes protocol NodeAdd during session $(\tau)$, then there exists a fixed value $z$ such that if $P_{new}$ reconstructs $z_i$ during session $(\tau)$, then $z_i = s$.*

**Privacy:** *The $t$-limited adversary cannot compute the shared secret $s$ after an execution of protocol NodeAdd, except with negligible probability.*

**Efficiency:** *The communication complexity for any instance of protocol NodeAdd is $d$-uniformly bounded.*

We obtain this by making three small modifications to our DKG.

- On receiving a Node-Add request, instead of running protocol Sh of HybridVSS for a random key, node $P_i$ reshares its current share $s_{i,\tau}$ and broadcasts the Node-Add request received. It then waits for $t$ other identical Node-Add requests before proceeding.
- Once a node $P_i$ receives $n - t - f$ ready messages for a decided set $\mathcal{Q}$, it Lagrange-interpolates $s_{i,d}$ for $P_d \in \mathcal{Q}$ for index *new* and provides subshare $s_{i,new}$ to node $P_{new}$ with commitments $V_\ell = \prod_{P_d \in \mathcal{Q}} ((C_{d,\tau})_{\ell,new})^{\lambda_d^{\mathcal{Q},new}}$ for $\ell \in [0, t]$.
- Node $P_{new}$, upon obtaining $t+1$ shares for same commitment vector $V_\ell$ for $\ell \in [0, t]$, interpolates them for index 0 to obtain its share $s_{new}$.

The main theorem for protocol NodeAdd is as follows.

**Theorem 6.1** *Assuming the hardness of the discrete-logarithm problem, protocol NodeAdd implements asynchronous verifiable node addition during a phase in the hybrid model for $n \geq 3t + 2f + 1$.*

We need to show liveness, consistency, privacy, and efficiency. The liveness and efficiency analysis exactly mirrors that of the DKG protocol. The worst-case message and communication complexities of the Renew protocol are $O(tdn^2(n+d))$ and $O(\kappa tdn^3(n+d))$ respectively. However, the consistency and privacy analyses are quite interesting.

**Consistency.** According to the consistency property of the reliable broadcast protocol, each of these nodes will complete the same set of $t+1$ sharings started for the Node-Add protocol. Assume that set $\mathcal{S}$ represents a set of nodes, which provide $t+1$ valid subshares to node $P_{new}$. Here, $s_{new}$ computed by node *new* is a valid share of the secret $s$ such that

$$\begin{aligned}
s_{new} &= \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} s_{i,new} \\
&= \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} ( \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},new} s_{i,d}) \\
&= \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},new} ( \sum_{P_i \in \mathcal{S}} \lambda_i^{\mathcal{S},0} s_{i,d}) \\
&= \sum_{P_d \in \mathcal{Q}} \lambda_d^{\mathcal{Q},new} s_d.
\end{aligned}$$

Node *new* can check the correctness of the received subshares using the commitment vector $V_\ell$ for $\ell \in [0,t]$ and protocol Rec at node *new* will reconstruct shared secret $s$. However, it is interesting to observe that it is not possible for a new node to generate its share, if we rely on subshares generated during protocol Renew instead of running a new DKG while adding the nodes.

**Privacy.** Here, it assumed that the addition of new nodes does not change the security threshold of the system. If the node to be added is honest, privacy can proven exactly in the same way as privacy in our DKG protocol. If it is dishonest, then it is one of the $t$ nodes which can be compromised by the adversary and knowing subshares of its share does not provide enough information to the adversary to extract the secret key.

### 6.3 Node Removal

This protocol involves removing a node from the system such that it should no longer be able to reconstruct the secret. Without modifying the shares for the other nodes, it is not possible to remove a node in the middle of a phase and we are restricted to removing it at the start of a new phase. To remove a node from the group involves simply not including it in the next share renewal protocol. An honest node should not carry out a node removal if that would invalidate the resilience bound $n \geq 3t + 2f + 1$.

### 6.4 Security Threshold and Crash-Limit Modification

Security threshold and crash-limit modification involves changing the threshold limit $t$ or the crash-limit $f$ of the system. For the same reason as node removal, it is not possible to modify the threshold and crash limits in the middle of a phase. With their lack of commutativity, we avoid direct threshold $t$ and crash-limit $f$ modifications. We modify $t$ and $f$ at the phase-change based on the all the node addition and removal requests confirmed during the previous phase. Nodes update their $t$ and $f$ values accordingly and start their HybridVSS instances with the updated parameter values. As a feature of the renewal protocol, the threshold value can be easily changed by just correctly changing the degrees of the resharing polynomials.

## 7 System Architecture

We design our DKG system as a deterministic state machine using the state machine replication approach [Lam78, Sch90]. As we aim at building an asynchronous distributed PKG for IBC as a representative DKG application, we are working on an object-oriented C++ implementation that uses the PBC library [Lyn09] for the underlying elliptic curve and finite field operations and a PKI infrastructure with DSA signatures based on GnuTLS [MFS$^+$09] for authentication and non-repudiation. We have already implemented a distributed PKG for Boneh-Franklin identity-based encryption in a partial-synchronous model [KG07], which is providing a coding framework for our DKG implementation.

#### System Design

In our deterministic state machine design, nodes moves from one state to another based on messages received. Messages are categorized into three types: operator messages, network messages and timer messages. Operator messages, which are of types `in` and `out`, define interactions between nodes and their operators. Network messages realize protocol flows between nodes. As we use a weak synchrony assumption to maintain liveness, we also include timer messages in the form of **start timer** and **stop timer**, which work according to the $delay(t)$ function described in §2.1.

#### Defence against DoS and Sybil Attacks

The distributed nature of DKG provides an inherent protection against DoS attacks and the inclusion of crashed nodes and network failure assumptions makes DoS attacks less feasible. Although leaders might become primary targets, we mitigate this issue with an efficient leader-changing mechanism. Further, as all valid communication is done over TLS links, nodes can easily reject messages arriving from non-system entities. Sybil attacks [Dou02] are not a major concern, as ad-hoc additions of nodes is not a feature of our system. Nodes are added using the group modification agreement protocol, which involves administrative interaction at each node.

# 8 Concluding Remarks

While working towards a realistic DKG architecture, we first investigated the differences between the partially synchronous and asynchronous communication models and observed that only the asynchronous communication model realistically fits existing Internet. We also incorporated crash-recoveries and network failures in the system along with the traditional Byzantine adversary and motivated their requirement in the proactive system models.

We defined a VSS scheme (HybridVSS) that works in our hybrid communication model. We then observed the requirement of a Byzantine agreement while implementing the DKG in the asynchronous communication setting and presented a leader-based system to achieve that in our DKG system. We are currently implementing our DKG protocol to prove its practicality and efficiency.

To achieve proactive security, we revisited our system model and suggested amendments to introduce the concept of phases into the asynchronous communication model and to maintain liveness and safety in the system. We presented share renewal and recovery mechanisms for our DKG protocol. We then observed the importance of group modification primitives for the long-term system sustainability and proposed protocols to achieve group modification agreement, node addition, node removal, and security-threshold and crash-limit modification. We plan to include these protocols in our DKG implementation such as distributed PKG for IBC.

## Acknowledgements

This work is supported by NSERC, MITACS, and a David R. Cheriton Graduate Scholarship. We thank the anonymous reviewers of ICDCS for their constructive feedback and David Schultz for interesting discussions and for providing a draft of their work in progress.

# References

[ADH08] I. Abraham, D. Dolev, and J. Y. Halpern, *An Almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience*, PODC'08, 2008, pp. 405–414.

[BC03] M. Backes and C. Cachin, *Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries*, DSN'03, 2003, pp. 37–46.

[BCS03] M. Backes, C. Cachin, and R. Strobl, *Proactive Secure Message Transmission in Asynchronous Networks*, PODC'03, 2003, pp. 223–232.

[BF01] D. Boneh and M. K. Franklin, *Identity-Based Encryption from the Weil Pairing*, Advances in Cryptology—CRYPTO'01, 2001, pp. 213–229.

[Bla79] G. R. Blakley, *Safeguarding Cryptographic Keys*, the National Computer Conference, 1979, pp. 313–317.

[BOCG93] M. Ben-Or, R. Canetti, and O. Goldreich, *Asynchronous Secure Computation*, STOC'93, 1993, pp. 52–61.

[Bra84] G. Bracha, *An Asynchronous [(n-1)/3]-Resilient Consensus Protocol*, PODC'84, 1984, pp. 154–162.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, *Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults*, FOCS'85, 1985, pp. 383–395.

[CKAS02] C. Cachin, K. Kursawe, A.Lysyanskaya, and R. Strobl, *Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems*, CCS'02, 2002, pp. 88–97.

[CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, *Secure and Efficient Asynchronous Broadcast Protocols*, Advances in Cryptology—CRYPTO'01, 2001, pp. 524–541.

[CKS00] C. Cachin, K. Kursawe, and V. Shoup, *Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography*, PODC'00, 2000, pp. 123–132.

[CL02] M. Castro and B. Liskov, *Practical Byzantine Fault Tolerance and Proactive Recovery*, ACM Trans. Comput. Syst. **20** (2002), no. 4, 398–461.

[CR93] R. Canetti and T. Rabin, *Fast asynchronous byzantine agreement with optimal resilience*, STOC'93, 1993, pp. 42–51.

[Des94] Y. G. Desmedt, *Threshold Cryptography*, European Transactions on Telecommunications **5** (1994), no. 4.

[DLS88] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, *Consensus in the Presence of Partial Synchrony*, Journal of ACM **35** (1988), no. 2, 288–323.

[Dou02] J. R. Douceur, *The Sybil Attack*, International Workshop on Peer-to-Peer Systems (IPTPS '02), 2002, pp. 251–260.

[DvOW92] W. Diffie, P.C. van Oorschot, and M. Wiener, *Authentication and authenticated key exchanges*, Designs, Codes and Cryptography **2** (1992), 107–125.

[Fel87] P. Feldman, *A Practical Scheme for Non-interactive Verifiable Secret Sharing*, FOCS'87, 1987, pp. 427–437.

[FLP85] M. J. Fischer, N. A. Lynch, and M. Paterson, *Impossibility of Distributed Consensus with One Faulty Process*, Journal of ACM **32** (1985), no. 2, 374–382.

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*, Journal of Cryptology **20** (2007), no. 1, 51–83.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest, *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, SIAM J. Comput. **17** (1988), no. 2, 281–308.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, *Proactive Secret Sharing Or: How to Cope With Perpetual Leakage*, Advances in Cryptology—CRYPTO'95, 1995, pp. 339–352.

[HT93] V. Hadzilacos and S. Toueg, *Fault-tolerant Broadcasts and Related Problems*, Distributed systems (2nd Ed.), ACM Press, 1993, pp. 97–145.

[KG07] A. Kate and I. Goldberg, *A Distributed Private-key Generator for Identity-Based Cryptography*, Tech. Report CACR 2007-33, University of Waterloo, 2007.

[KKA03] A. Khalili, J. Katz, and W. Arbaugh, *Toward Secure Key Distribution in Truly Ad-Hoc Networks*, IEEE Workshop on Security and Assurance in Ad-Hoc Networks 2003, 2003, pp. 342–346.

[KZG07] A. Kate, G. M. Zaverucha, and I. Goldberg, *Pairing-Based Onion Routing*, 7th Privacy Enhancing Technologies Symposium (PET), 2007, pp. 95–112.

[Lam78] L. Lamport, *Time, clocks, and the ordering of events in a distributed system*, Commun. ACM **21** (1978), no. 7, 558–565.

[Lyn09] B. Lynn, *PBC Library*, `http://crypto.stanford.edu/pbc/`, 2009, Accessed April 2009.

[MFS+09] N. Mavroyanopoulos, F. Fiorina, T. Schulz, A. McDonald, and S. Josefsson, *The GNU Transport Layer Security Library*, `http://www.gnu.org/software/gnutls/`, 2009, Accessed April 2009.

[Nie02] J. B. Nielsen, *A Threshold Pseudorandom Function Construction and Its Applications*, Advances in Cryptology—CRYPTO'02, 2002, pp. 401–416.

[NPR99] M. Naor, B. Pinkas, and O. Reingold, *Distributed Pseudo-random Functions and KDCs*, Advances in Cryptology—EUROCRYPT'99, 1999, pp. 327–346.

[OY91] R. Ostrovsky and M. Yung, *How to Withstand Mobile Virus Attacks (Ext. Abstract)*, PODC'91, 1991, pp. 51–59.

[PCR08] A. Patra, A. Choudhary, and C. Pandu Rangan, *Efficient Asynchronous Verifiable Secret Sharing and Byzantine Agreement with Optimal Resilience*, Cryptology ePrint: 2008/424, 2008.

[Ped91] T. P. Pedersen, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, Advances in Cryptology—CRYPTO'91, 1991, pp. 129–140.

[Sch90] F. B. Schneider, *Implementing fault-tolerant services using the state machine approach: A tutorial*, ACM Comput. Surv. **22** (1990), no. 4, 299–319.

[Sha79] A. Shamir, *How to Share a Secret*, Commun. ACM **22** (1979), no. 11, 612–613.

[SLL08] D. A. Schultz, B. Liskov, and M. Liskov, *Mobile Proactive Secret Sharing*, PODC'08, 2008, (Extended Draft), p. 458.

[ZSvR05] L. Zhou, F. B. Schneider, and R. van Renesse, *APSS: Proactive Secret Sharing in Asynchronous Systems*, ACM Trans. Inf. Syst. Secur. **8** (2005), no. 3, 259–286.