Distributed Key Generation and Its Applications

by

Aniket Pundlik Kate

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2010

Abstract

Numerous cryptographic applications require a trusted authority to hold a secret. With a plethora of malicious attacks over the Internet, however, it is difficult to establish and maintain such an authority in online systems. Secret-sharing schemes attempt to solve this problem by distributing the required trust to hold and use the secret over multiple servers; however, they still require a trusted *dealer* to choose and share the secret, and have problems related to single points of failure and key escrow. A distributed key generation (DKG) scheme overcomes these hurdles by removing the requirement of a dealer in secret sharing. A (threshold) DKG scheme achieves this using a complete distribution of the trust among a number of servers such that any subset of servers of size greater than a given threshold can reveal or use the shared secret, while any smaller subset cannot. In this thesis, we make contributions to DKG in the computational security setting and describe three applications of it.

We first define a constant-size commitment scheme for univariate polynomials over finite fields and use it to reduce the size of broadcasts required for DKG protocols in the synchronous communication model by a linear factor. Further, we observe that the existing (synchronous) DKG protocols do not provide a liveness guarantee over the Internet and design the first DKG protocol for use over the Internet. Observing the necessity of long-term stability, we then present proactive security and group modification protocols for our DKG system. We also demonstrate the practicality of our DKG protocol over the Internet by testing our implementation over PlanetLab.

For the applications, we use our DKG protocol to define IND-ID-CCA secure distributed private-key generators (PKGs) for three important identity-based encryption (IBE) schemes: Boneh and Franklin's BF-IBE, Sakai and Kasahara's SK-IBE, and Boneh and Boyen's BB_1-IBE. These IBE schemes cover all three important IBE frameworks: full-domain-hash IBEs, exponent-inversion IBEs and commutative-blinding IBEs respectively, and our distributed PKG constructions can easily be modified for other IBE schemes in these frameworks. As the second application, we use our distributed PKG for BF-IBE to define an onion routing circuit construction mechanism in the identity-based setting, which solves the scalability problem in single-pass onion routing circuit construction without hampering forward secrecy. As the final application, we use our DKG implementation to design a threshold signature architecture for quorum-based distributed hash tables and use it to define two robust communication protocols in these peer-to-peer systems.

Acknowledgements

First, a very special thanks is due to my supervisor, Ian Goldberg. Along with being an extraordinary researcher and an enthusiastic collaborator, he has been a devoted advisor and a dear friend. I feel very fortunate for having had the chance to work closely with him.

My interaction with Ian has deeply affected my approach to research. His striking clarity of thought and amazing listening ability have been vital to providing structure to my unorganized ideas. Furthermore, his predilection for practicality and implementations in research has been critical to whatever industrial usefulness of my research work has. I am also thankful to him for his patience while dealing with my grammatical errors, and for keeping me on the track which resulted in a coherent thesis rather than a set of stapled papers.

Next I wish to thank my collaborators on the results that make up this thesis. It has been my pleasure to work with them. The PolyCommit commitment protocol (Chapter 3) and pairing-based onion routing (Chapter 7) is joint work with Greg Zaverucha, while the robust communication protocols for DHTs (Chapter 9) is joint work with Maxwell Young and Martin Karsten. Ian is also a collaborator in the above and the rest of my thesis work.

I am also thankful to my other committee members Urs Hengartner, Alfred Menezes, Michael Reiter and Doug Stinson for their motivational discussions and suggestions throughout my studies at Waterloo and for helping me to improve my thesis at the end. A special thanks to Urs for supporting me during my first two terms in the program.

At Waterloo, I have been also fortunate to work with an excellent set of colleagues including Jason, Atefeh, Berkant, Jiang, Greg, Mridul, Joel, Jerry, Kevin, Ge, Sumair, Chris, Marcio, Maura, Jeremy, Max, Sanjeet, Koray, Femi, Mehrdad, Ryan, Can, Anne, and Jalaj. I should especially mention Greg Zaverucha here. A collaboration we started four years back in a tiny student office has been overwhelmingly successful and continues to grow. I am also thankful to Atefeh Mashatan for her knowledgeable and thoughtful suggestions in administrative matters when I needed them the most. Finally, the CrySP lab would not have been fun without the alert and cheerful mind of Jeremy Clark. I am thankful to him for the smiles he spread.

During the last four enriching years at Waterloo, I met and made numerous friends. It is certainly hard to choose only a few of them here. The time I spent with my friends remains precious to me. However, my life in Waterloo would not have been complete without a managerial experience at my residential co-operative with Sudipto, stimulating discussions in the Udai meetings, Tim Horton's double-double coffee with Sarvagya, being a subject for peculiar experiments of the friends from Optometry, childish and cheerful conversations with Geeta, fierce discussions in Mafia games, and sparring competitions in the Karate classes. My housemates during last four years (Avisek, Kammy, Shanee, Sourav, Sarah, Sudipto, Syamantak and Arijit) are among the people closest to me in Canada. I also thank my various partners in sports and other activities for keeping me reinvigorated during my academic work. A special thanks to Sarvagya, Darya and Jeremy for reading some parts of my thesis.

I must also thank Bruce Schneier for his book *Applied Cryptography* which introduced me to the fascinating world of cryptography and security during my undergraduate studies and made me its devotee.

Finally, I want thank my grandmother for taking care of me during my childhood, my parents for their unconditional love and encouraging me to pursue my own dreams, my brother for imparting a competitive spirit and lately my sister-in-law for those cheerful and lively chats. I owe every bit of my achievements to my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Malicious behaviour was never so prevalent over the Internet. The abundance of valuable online information about individuals, organizations and governments has made attacks on the Internet-based services a very successful venture for organized crime, terrorist outfits and even government agencies. With their existing scale and substantial economic advantages, there is no perceivable end to these malicious activities over the Internet. As a result, it is difficult to establish and maintain a trusted authority over the Internet, which is indispensable for many Internet-based cryptographic applications. A naive approach can be to replicate the information (or the computation) to multiple servers so that a failure at a few servers does not shut down the corresponding service. However, this is not suitable against a malicious attack where an adversary obtains or modifies the secret information by compromising any one of these replicated servers.

Threshold cryptography [DF89] solves this problem by fault-tolerantly distributing the information among a group of cooperating servers under the reasonable assumptions that the majority of servers in the group remains trustworthy. The concept of *secret sharing* [Bla79, Sha79] forms the basis of threshold cryptography. It allows a *dealer* to distribute a piece of secret information among a group of servers such that no subset of corrupt servers (smaller than or equal to a given threshold) can figure out what the secret is, even if they cooperate; moreover, when it becomes necessary to reconstruct the secret information, a number of servers larger than the above threshold can always do it. In threshold cryptography, secret sharing is used to define function sharing such that a highly sensitive cryptographic operation (*e.g.*, decryption or signing) is performed by a group of cooperating servers. Here, any subset of servers of size at most the threshold is unable to either perform the operation by themselves or prevent the other *honest* servers from performing the operation.

In secret sharing, a dealer knows the secret it shares. It has to be therefore trusted not to crash until the completion of a secret sharing instance is guaranteed, and to erase or at least not to divulge the secret before it is reconstructed. In many cryptosystems, we require complete distribution of the trust and no party (not even a dealer) should be in the sole possession of the secret. The concept of *distributed key generation* (DKG) [Ped91b] satisfies this requirement. In a DKG protocol, a shared secret is collectively generated in a group in a completely distributed way such that any subset of size greater than a threshold can reveal or use the shared secret, while smaller subsets do not have any knowledge about it. Importantly, there is no dealer; each node in the group runs an instance of a secret sharing scheme and computes its final share by adding the shares it received from the successful secret sharing instances.

The concept of DKG has numerous applications in cryptography. In symmetric-key cryptography, DKG is used to design distributed key distribution centres [NPR99]. Here, a group of servers jointly realize the function of a key distribution centre, which generates and provides encryption keys for secure conferences to clients. In public-key cryptography (PKC), DKG is essential for dealerless threshold public-key decryption and signature schemes [DF89] and for distributed private-key generation in identity-based cryptography (IBC) [BF01]. In a threshold decryption 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 have to combine their partial signatures to sign a message. In distributed key distribution centres, threshold decryption and signature schemes, DKG tackles the problem of *single point of failure*. In IBC, it also mitigates the *key escrow* issue, when it is impractical 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 necessary 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]. DKG is also an important primitive in distributed pseudo-random functions [NPR99], which are useful in distributed coin tossing algorithms [CKS00] and random oracles [Nie02].

Although various theoretical aspects of DKG have been thoroughly researched for the last two decades, the systems aspects have been largely ignored. Existing DKG protocols rely on assumptions like synchronous communication (bounded communication delay, with known bounds) or ask for a reliable broadcast channel. As these prerequisites are not readily available over the Internet, the existing DKG protocols are not suitable for the Internet-based applications and there is no DKG available or used in practice yet.

> This thesis designs and implements a practical DKG protocol for use over the Internet and describes its application in IBC, onion routing circuit construction and robust communication in peer-to-peer systems.

1.1 Contributions

In this section, we present an informal description of our contributions, which are thoroughly discussed in the rest of the thesis. Our contributions fall into two main categories.

The first category consists of our work on the core concept of (polynomial-based) secret sharing and DKG protocols. We define a constant-size commitment scheme for univariate polynomials over finite fields and utilize it to reduce the communication cost for the secret sharing and DKG protocols. We then present the first DKG protocol for use over the Internet and test its implementation over the PlanetLab platform. Observing the necessity of long-term stability, we also consider proactive security and group modification primitives for our DKG protocol.

The second category explores applications of DKG in the discrete logarithm (DLog) setting. We observe the necessity of distributing PKG for any Internet-scale application of IBC and define distributed PKGs for all three important identity-based encryption (IBE) frameworks. We then use one of the above distributed PKGs to define an onion routing circuit construction protocol in the identity-based setting, which simultaneously solves the efficiency and scalability problems in onion routing circuit construction. Finally, we utilize a threshold signature scheme with our DKG protocol to define two robust communication protocols for peer-to-peer systems based on distributed hash tables (DHTs) secure against a malicious (*Byzantine*) adversary.

Constant-Size Commitments To Polynomials. All known verifiable secret sharing (VSS) schemes [CGMA85] use commitments that are at least linear in the group size; they use commitments to the polynomial coefficients or the evaluations to commit to the shared polynomial. This results in a linear-size broadcast for the VSS protocols in the synchronous communication setting.

In Chapter 3, we introduce and formally define the concept of polynomial commitment schemes, and provide an efficient construction PolyCommit for univariate polynomials [KZG10]. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string such that she can later open the commitment to evaluations of the committed polynomial. A verifier can verify the correctness of these openings (evaluations) with respect to the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, as we discuss above, the sizes of their commitments are linear in the degree of the committed polynomial. In contrast, a polynomial commitment in our PolyCommit scheme is a single element (constant size). Furthermore, the overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, PolyCommit is a useful tool to reduce the communication cost in cryptographic protocols. In the context of this thesis, we apply our constant-size PolyCommit scheme to the Feldman VSS scheme [Fel87] to reduce its broadcast size from a linear-size vector to a single element. Finally, we convert the new VSS (eVSS) scheme to the joint-Feldman DKG (JF-DKG) protocol [GJKR07] in the synchronous setting using a non-interactive zero knowledge proof technique to generate its efficient version eJF-DKG.

Distributed Key Generation (DKG). DKG has never been examined outside of the synchronous setting. In Chapter 4, we present the first realistic DKG implementation for use over the Internet [KG09, KG10a]. We propose a practical hybrid system model that combines the malicious (Byzantine) adversary with crash recovery and network failures in the asynchronous communication setting (no bounds over the communication delays), and discuss its applicability over the Internet. We modify the AVSS protocol [CKAS02] to work in our hybrid system model, which results in our HybridVSS scheme. We observe the necessity of *Byzantine agreement* [LSP82] in an asynchronous DKG protocol to agree on the set of VSS instances to be included in DKG and analyze the difficulty of using a randomized Byzantine agreement for it. Using our VSS scheme in the hybrid model and a leader-based agreement protocol, we then design a provably secure DKG (HybridDKG) protocol for use over the Internet. To establish the efficiency and the reliability of HybridDKG, we develop a C++ implementation, and analyze our protocol implementation on the PlanetLab platform [PACR03].

For long-lived distributed secrets the above threshold-based protection is not sufficient. In Chapter 5, we describe a proactive share renewal primitive for HybridDKG such that shares are periodically renewed (without changing the secret) in a way that information gained by the adversary in a time period is useless for attacking the secret, once the shares are renewed. We also define a share recovery protocol for a node that recovers from a crash. Along with these traditional proactive security measures we observe the importance of group modification primitives in our DKG protocol and define protocols to add/remove nodes and modify the threshold values.

Our HybridDKG protocol uses DLog commitments and consequently does not guarantee uniform randomness of the shared secret key [GJKR07]. However, we observe that in the random oracle model, using non-interactive zero-knowledge proofs of knowledge based on the Fiat-Shamir methodology [FS86], it is possible to achieve uniform randomness in HybridDKG. In this scheme, the DLog commitments are initially replaced by Pedersen commitments [Ped91b]; the DLog commitments are introduced only at the end of the protocol to obtain the required public key. The zero-knowledge proofs are used to show that the DLog and Pedersen commitments both commit to the same values.

Note that our PolyCommit scheme does not seem to work with the bivariate polynomials that are used in all VSS and DKG schemes in the asynchronous setting. Therefore, we are not able to use the efficient PolyCommit commitment scheme in our HybridVSS and HybridDKG schemes, and they have a linear communication gap between the message (number of messages transferred) complexity and bit (number of bits transferred) complexity.

Distributed Private Key Generators (Distributed PKGs). An identity-based encryption (IBE) scheme can greatly reduce the complexity of sending encrypted messages over the Internet. However, an IBE scheme necessarily requires a PKG, which can create private keys for clients and so can passively eavesdrop on all encrypted communications. Although a distributed PKG has been suggested as a way to mitigate this key escrow problem for Boneh and Franklin's IBE scheme, the security of this distributed protocol has not been proven and the proposed solution does not work over the asynchronous Internet. Further, a distributed PKG has not been considered for any other IBE scheme.

As the first application of our DKG protocols, in Chapter 6, we design distributed PKG setup and private key extraction protocols for three important IBE schemes: Boneh and Franklin's BF-IBE [BF01], Sakai and Kasahara's SK-IBE [SK03], and Boneh and Boyen's BB_1-IBE [Boy08, BM07]. We give special attention to the applicability of our protocols to all possible types of bilinear pairings defined by Galbraith *et al.* [GPS08], and prove their adaptive chosen ciphertext security (IND-ID-CCA) in the random oracle model against a Byzantine adversary. We also perform a comparative analysis of these protocols and present recommendations for their use.

Note that our three schemes cover all three important IBE frameworks: full-domain-hash IBEs, exponent-inversion IBEs, and commutative-blinding IBEs as defined by Boyen [Boy08]. Futhermore, as they can be easily converted to the other schemes in their framework category, our work proposes distributed PKGs for all practical IBE schemes.

Pairing-Based Onion Routing. Over the last few years onion routing networks have emerged as the most successful solutions for anonymous web browsing. The most popular existing onion routing network, Tor, is still far from optimal with its circuit construction delays and scalability issues.

As the second application of our DKG protocols, in Chapter 7, we use the above distributed PKG for BF-IBE to define new circuit construction protocols for onion routing anonymity networks [KZG07, KZG09]. We define a provably secure anonymous key agreement scheme in the BF-IBE setting, and use it to design new onion routing circuit constructions (PB-OR and λ-pass PB-OR). These constructions, based on a user's selection, offer immediate or eventual forward secrecy at each node in a circuit and require significantly less computation and communication than the telescoping mechanism used by Tor [DMS04]. Further, the use of an identity-based infrastructure also leads to a reduction in the required amount of authenticated directory information. Therefore, our constructions provide practical ways to allow onion routing anonymity networks to scale gracefully.

We also present compact message formats for onion routing circuit construction [KG10b] using the Sphinx methodology [DG09] developed for mixes. With this, we significantly compress the circuit construction messages for the above PB-OR protocol. Our new circuit construction is also secure in the universal composability (UC) framework [Can01], a property that was missing from the original construction. Futhermore, we compare the message size and the efficiency of the new Sphinx-based scheme with the original PB-OR scheme and observe that the Sphinx-based construction achieves a significant reduction in the circuit construction message lengths.

Robust DHT. Several analytical results exist on distributed hash tables (DHTs) which can tolerate Byzantine faults. Unfortunately, these results incur significant communication cost in order to achieve message routing. For instance, for a DHT system of η nodes, a naive routing scheme that has been used previously has $O(\log^3 \eta)$ message complexity, which, in general, is impractical for real-world applications. The previously best known protocol achieves an expected $O(\log^2 \eta)$ messages [SY08]. However, the protocol suffers from large hidden constants and setup costs [YKGK10].

In Chapter 9, using the concept of threshold digital signatures over our HybridDKG scheme, we obtain two robust communication protocols. Both of these protocols asymptotically improve the communication costs of previous solutions against a computationally bounded Byzantine adversary [YKGK10]. In comparison, our first protocol is deterministic and achieves $O(\log^2 \eta)$ message complexity, and our second protocol is randomized and achieves $O(\log \eta)$ message complexity in expectation. The hidden constants and setup costs for our protocols are small, and no trusted third party is required. In the end, we present results from microbenchmarks conducted over the PlanetLab platform, which show that our protocols are practical for deployment under significant levels of churn and adversarial behaviour.

1.2 Organization

In the next chapter (Chapter 2), we provide the necessary background information for the topics discussed in the thesis. The rest of the thesis is divided into three parts: distributed key generation, its cryptographic applications, and its implementation and corresponding system-level application to DHTs.

In Part I, we present our work on VSS and DKG protocols. Chapter 3 describes our constant-size PolyCommit commitment scheme for univariate polynomials over finite fields and uses it to define the efficient eVSS and eJF-DKG schemes having a constant-size broadcast per VSS instance. In Chapter 4, we propose and prove our HybridDKG scheme to solve the problem of distributed key generation in the DLog setting for use over the Internet. We present proactive security and group modification protocols for HybridDKG in Chapter 5.

In Part II, we discuss the cryptographic applications of DKG. In Chapter 6, we design distributed PKG protocols for all three important IBE frameworks: full-domain-hash IBEs, exponent-inversion IBEs and commutative-blinding IBEs, and also perform a comparative study among them. In Chapter 7, we use one of the designed distributed PKGs to construct onion routing circuits in the identity-based setting and include two extensions: a multi-pass construction achieving better forward secrecy and a compact UC-secure circuit construction.

In Part III, we concentrate on the implementation aspects of HybridDKG. In Chapter 8, we describe the design and implementation of our HybridDKG protocol, and discuss and test the system-level optimizations that we apply. We also include a detailed account on our experiments over the PlanetLab platform. In Chapter 9, we use our implementation to realize two practical robust communication protocols for DHTs against a computationally bounded Byzantine adversary. Finally, in Chapter 10 we conclude the thesis and present some possible future work.

Chapter 2

Background

In this chapter, we prepare the background for the rest of the thesis. In Section 2.1, we define our notation. Section 2.2 and Section 2.3 describe the concepts of verifiable secret sharing and distributed key generation, respectively. In Section 2.4, we define the cryptographic assumptions that we use throughout this thesis; assumptions specific to particular topics are included in their respective chapters. In Section 2.5, we describe the concept of bilinear pairings. In the end, we overview homomorphic commitment schemes (Section 2.6) and the non-interactive zero-knowledge proofs in the random oracle model (Section 2.7) that we frequently use in the thesis. An approach here is to provide an overview of the above concepts rather than to discuss them thoroughly; thus, readers seasoned in these topics can skip this chapter for now, and as required, refer to its sections with the backward references provided.

2.1 Notation

We work in the computational security setting, where κ denotes the security level of the system, in bits. That is, our probabilistic polynomial time (PPT) adversary algorithm $\mathcal{A}$ has to perform 2^κ computations to break the security of a cryptosystem. $\mathbb{G}$ represents a multiplicative group of prime order p. The size of the prime p depends upon κ; it is chosen to achieve κ bits security under the computational hardness assumption of the system and changes with the assumption used. g and h denote elements of $\mathbb{G}$. In some cryptosystems, we need two additional groups of the same order p as $\mathbb{G}$; we represent them as $\hat{\mathbb{G}}$ and $\mathbb{G}_T$. All these groups are represented in a multiplicative form. In the cryptographic practice, multiplicative subgroups of finite fields [MOV97, Chapter 3] and elliptic curve groups defined over finite fields [HMV04] are the two sources most commonly used to generate these prime-order groups.

Definition 2.1. Negligible Function. *A function $\epsilon(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is called* negligible *if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$.*

In the remainder of the thesis, $\epsilon(\cdot)$ will always denote a negligible function.

In our distributed settings, the adversary is always bounded to be able to corrupt or control a threshold number of nodes in a group; we use t to denote this threshold. n always represents the total number of nodes in the group and f represents the number of crashes and link failures. We use letters P, Q and U to denote individual nodes (peers) in the group, while $\mathcal{Q}$ and Q are used denote (sub)sets of the nodes. s and z are used to denote secrets in the system, ϕ and ψ represent the polynomials over $\mathbb{F}_p$, and H denotes a cryptographic hash function. In our security proofs, h is used for hash values, and in context does not conflict with the use of h as an element of $\mathbb{G}$. Finally, T represents the time since a protocol instance is started.

2.2 Verifiable Secret Sharing (VSS)

The notion of secret sharing was introduced independently by Shamir [Sha79] and Blakley [Bla79] in 1979. Since then, it has remained an important topic in cryptographic research.

Definition 2.2. **$(n, t+\delta, t)$-Secret Sharing**. *For integers n, t and δ such that $n \geq t+\delta > t \geq 0$, an $(n, t+\delta, t)$-*secret sharing *scheme is a protocol used by a dealer to share a secret s among a set of n nodes in such a way that any subset of $t+\delta$ or more nodes can compute the secret s, but subsets of size t or fewer cannot.*

A secret sharing scheme with $\delta = 1$ is called a *threshold* secret sharing scheme, and for $\delta > 1$, it is called a *ramp* secret sharing scheme. In this work, we concentrate on threshold secret sharing and denote it as (n, t)*-secret sharing* instead of $(n, t+1, t)$-secret sharing. Note that all polynomial-based threshold secret sharing schemes can easily be converted to ramp secret sharing schemes [Sti05].

In secret sharing, nodes may need a procedure to verify the correctness of the dealt values in order to prevent malicious behaviour by the dealer. To solve this problem, Chor *et al.* [CGMA85] introduced verifiability in secret sharing, which led to the concept of *verifiable secret sharing* (VSS).

Definition 2.3. **(n, t)-Verifiable Secret Sharing (VSS)**. *An (n, t)-VSS scheme consists of two phases: the* sharing *(Sh) phase and the* reconstruction *(Rec) phase.*

Sh phase. *A dealer P_d distributes a secret $s \in \mathcal{K}$ among n nodes, where $\mathcal{K}$ is a sufficiently large key space. At the end of the Sh phase, each honest node P_i holds a share s_i of the distributed secret s.*

Rec phase. *In this phase, each node P_i broadcasts its secret share s'_i and a reconstruction function is applied in order to compute the secret $s = \mathsf{Rec}(s'_1, s'_2, \ldots, s'_n)$ or output $\perp$ indicating that P_d is malicious. For honest nodes $s'_i = s_i$, while for malicious nodes s'_i may be different from s_i or even absent.*

It has two security requirements: Secrecy and Correctness.

Secrecy (VSS-wS). *A* t-limited *adversary who can compromise t nodes cannot compute s during the Sh phase.*

Correctness (VSS-C). *The reconstructed value z should be equal to the shared secret s or every honest node concludes that the dealer is malicious by outputing $\perp$.*

In the thesis, we consider VSS schemes in the computational complexity setting. Here, any malicious behaviour by P_d is caught by the honest nodes in the Sh phase itself and the VSS-C property simplifies to the following: the reconstructed value z should be equal to the shared secret s.

Further, many VSS applications avoid participation by all parties during the Rec phase. It is required that broadcasts from any $t+1$ honest nodes (or any $2t+1$ nodes) is sufficient to reconstruct s. Therefore, along with secrecy and correctness, we mandate the correctness property that we refer as *strong correctness* requirement.

Strong Correctness (VSS-SC). The same unique value s is reconstructed regardless of the subset of nodes (of size greater than $2t$) chosen by the adversary in the Rec algorithm.

Further, some VSS schemes achieve a stronger secrecy guarantee.

Strong Secrecy (VSS-S). The adversary who can compromise t nodes does not have any more information about s except what is implied by the public parameters.

Finally, VSS has remained an important area of cryptographic research for the last two decades [Fel87, DF89, Ped91a, Ped91b, HJKY95, CGJ$^+$99, FMY99, GJKR99, JL00, CKAS02, GJKR03, AF04, GJKR07]. In Chapters 3, 4 and 5, we make new contributions to it, which have appeared in [KG09, KZG10].

2.3 Distributed Key Generation (DKG)

Pedersen [Ped91b] introduced the concept of distributed key generation (DKG) and developed a DKG protocol, where each node runs a VSS instance and distributed shares are added at the end to generate a combined shared secret *without a dealer*. Unlike VSS, where a dealer chooses a secret and distributes its shares among the nodes, DKG requires no trusted party.

Definition 2.4. (n, t)**-Distributed Key Generation (DKG)**. *An* (n, t)*-DKG scheme consists of two phases: the* sharing *(*Sh*) phase and the* reconstruction *(*Rec*) phase.*

Sh phase. *Every node* P_i *distributes a secret* $z_i \in \mathcal{K}$ *among* n *nodes, where* $\mathcal{K}$ *is a sufficiently large additive cyclic group. At the end of the* Sh *phase, each honest node* P_i *holds a share* s_i *of the distributed secret* s*, which is a pre-decided linear combination of the shared* z_i *values.*

Rec phase. *Each node* P_i *broadcasts its secret share* s'_i *and a reconstruction function is applied in order to compute the secret* $s = \mathsf{Rec}(s'_1, s'_2, \ldots, s'_n)$*. For honest nodes* $s'_i = s_i$*, while for malicious nodes* s'_i *may be different from* s_i *or even absent.*

As discussed in the introduction, DKG has numerous application in cryptography. In the thesis, we concentrate on threshold cryptosystems in the DLog setting. In this setting, having a cyclic group $\mathbb{G}$ of prime order p, a DKG protocol generates secret sharing of a secret $s \in \mathbb{Z}_p$, and publishes $Y = (g, g^s)$ for $g \in_R \mathbb{G}$ as the corresponding public key. Gennaro *et al.* [GJKR07, Section 4.1] suggested the following secrecy and correctness requirements for an (n, t)-DKG protocol in the DLog setting:

Correctness (DKG-C).

1. There is an efficient algorithm that on input shares from $2t + 1$ nodes and the public information produced by the DKG protocol outputs the same unique value s, even if up to t shares are submitted by malicious nodes.[1]
2. At the end of Sh phase, all honest nodes have the same value of the public key $Y = g^s$, where s is the unique secret guaranteed above.
3. s and Y are uniformly distributed in $\mathbb{Z}_n$ and $\mathbb{G}$, respectively.

Secrecy (DKG-S). No information about s can be learned by the adversary except for what is implied by $Y = g^s$.

[1]Note that this is the stronger version, which they define in the latter part of [GJKR07, Section 4.1].

Gennaro *et al.* [GJKR99] found that DKGs based on the Feldman VSS methodology do not guarantee uniform randomness of the shared secret key or DKG-S. Further, they observed that the use of digital signatures or other slight modifications do not provide any additional security to the Pedersen DKG [Ped91b] and presented a simplification using just the original Feldman VSS called the Joint Feldman DKG (JF-DKG) as the representative scheme. They then defined a new DKG protocol combining discrete logarithm and Pedersen commitments [Ped91b], which guarantees the uniform randomness property by increasing the *latency* (number of communication rounds) of the DKG protocol by one.

In [GJKR03], the same set of authors observed that the Pedersen DKG and JF-DKG produce hard instances of the DLog problem, which may be sufficient for the security of some threshold cryptographic schemes. Although a reduction in their JF-DKG security proof is not tight, as they discussed in [GJKR07], an elliptic-curve implementation of JF-DKG with appropriately increased key sizes is still faster than the modification they suggested in [GJKR99]. In this weaker version of DKG, the third correctness property is absent and the secrecy requirement weakens to the following:

Weak Correctness (DKG-wC).

1. There is an efficient algorithm that on input shares from $2t + 1$ nodes and the public information produced by the DKG protocol outputs the same unique value s, even if up to t shares are submitted by malicious nodes.
2. At the end of Sh phase, all honest nodes have the same value of public key $Y = g^s$, where s is the unique secret guaranteed above.

Weak Secrecy (DKG-wS). The adversary with t shares and the public key $Y = g^s$ cannot compute the secret s.

Knowing this efficiency and secrecy tradeoff, we define two versions of our DKG constructions: an efficient DKG with weak correctness and weak secrecy, and a DKG with uniform randomness of the shared secret and strong secrecy.

It is important to discuss the relationship between the various secrecy and correctness notions that we present in the above two sections. For secrecy, VSS-wS and VSS-S are equivalent to DKG-wS and DKG-S respectively. For correctness, VSS-SC is included in both DKG-wS and DKG-S; the weaker VSS-C is a property common only in the unconditional VSS protocols.

Note that the above properties may require some cryptographic assumptions and generally have a negligible probability of error attached to them.

2.4 Cryptographic Assumptions

All of our adversaries are PPT algorithms with respect to the security parameter κ. In other words, they are computationally bounded by κ, and have to do 2^κ computation to break the security of any of the protocols except with a negligible probability.

Our protocols work in a distributed setting. Our adversaries are t-limited *Byzantine* adversaries, who compromise up to t nodes in the system and make them behave arbitrarily. Further they are *rushing* and can wait for the messages of the uncorrupted (honest) players to be transmitted, then decide on their computations and communications, and still get their messages delivered to the honest parties on time. They are also of a *static* nature and have to choose their t compromisable nodes before a protocol run.

Our distributed cryptographic protocols in this thesis and most of the distributed cryptographic protocols in the literature are not considered secure against an *adaptive* adversary that may choose its t compromisable nodes as a protocol is getting executed. As elaborated in [GJKR07, Section 4.4], this is only because their (simulation-based) security proofs do not go through when the adversary can corrupt nodes adaptively. In [Fel87, Section 9.3], Feldman claimed that his VSS protocol is also secure against adaptive adversaries even though his simulation-based security proof did not work out. Canetti *et al.* [CGJ$^+$99] presented a distributed protocol methodology that is provably secure against adaptive adversaries using interactive zero-knowledge proofs and a proof technique that may rewind the adversary a polynomial number of times. However, their methodology in general adds at least two more communication rounds to a protocol, which can severely deteriorate the system performance. On the other hand, all of the protocols in the literature that are proven secure only against a static adversary have remained unattacked by an adaptive adversary for the last 22 years. Gaining some confidence from this fact and giving importance to efficiency, we stick to protocols provably secure only against a static adversary in our work.

The discrete logarithm (DLog) assumption [MOV97, Sec. 3.6] is one of the fundamental assumptions in PKC and we extensively use it in our work. It relies on the hardness of computing an exponent a given two group elements g and g^a in some group $\mathbb{G}$.

Definition 2.5. Discrete Logarithm (DLog) Assumption. *Given a generator g of a multiplicative group $\mathbb{G}$ of prime order p and $a \in \mathbb{Z}_p^*$, for every adversary $\mathcal{A}_{\mathsf{DLog}}$, $\Pr[\mathcal{A}_{\mathsf{DLog}}(g, g^a) = a] = \epsilon(\kappa)$.*

Diffie and Hellman [DH76] and most of the DLog systems afterwards used the following assumption for their constructions, which is called the computational Diffie-Hellman (CDH) assumption.

Definition 2.6. Computational Diffie-Hellman (CDH) Assumption. *Given a generator g of a multiplicative group $\mathbb{G}$ of prime order p and $g^a, g^b \in \mathbb{G}$ for unknown $a, b \in_R \mathbb{Z}_p$, for every adversary $\mathcal{A}_{\mathsf{CDH}}$,* $\Pr[\mathcal{A}_{\mathsf{CDH}}(g, g^a, g^b) = g^{ab}] = \epsilon(\kappa)$.

Along with the CDH assumption, we use its decision version: decision Diffie-Hellman (DDH) assumption [Bon98].

Definition 2.7. Decision Diffie-Hellman (DDH) Assumption. *Given a generator g of a multiplicative group $\mathbb{G}$ of prime order p and $g^a, g^b, h \in_R \mathbb{G}$ for unknown $a, b \in_R \mathbb{Z}_p$, for every adversary $\mathcal{A}_{\mathsf{DDH}}$,* $\Pr[\big(\mathcal{A}_{\mathsf{DDH}}(g, g^a, g^b, g^{ab}) = \mathit{true}\big) - \big(\mathcal{A}_{\mathsf{DDH}}(g, g^a, g^b, h) = \mathit{true}\big)] = \epsilon(\kappa)$.

We also use another related assumption at multiple places. Mitsunari, Sakai and Kasahara [MSK02] introduced the *weak Diffie-Hellman assumption*, which was renamed the t-Diffie-Hellman Inversion (t-DHI) assumption by Boneh and Boyen [BB04a] as this assumption is stronger than the above CDH assumption, especially for large values of t.[2]

Definition 2.8. t-Diffie-Hellman Inversion (t-DHI) Assumption. *Let $\alpha \in_R \mathbb{Z}_p^*$. Given as input a $(t+1)$-tuple $\langle g, g^\alpha, \ldots, g^{\alpha^t} \rangle \in \mathbb{G}^{t+1}$, for every PPT algorithm $\mathcal{A}_{t\text{-}\mathsf{DHI}}$, the probability* $\Pr[\mathcal{A}_{t\text{-}\mathsf{DHI}}(g, g^\alpha, \ldots, g^{\alpha^t}) = g^{\frac{1}{\alpha}}] = \epsilon(\kappa)$.

It is easy to see that the above t-DHI assumption is equivalant to the assumption that probability $Pr[\mathcal{A}_{t\text{-}\mathsf{DHI}}(g, g^\alpha, \ldots, g^{\alpha^t}) = g^{\alpha^{t+1}}] = \epsilon(\kappa)$ [BBG05]. See Cheon [Che06] for a security analysis.

2.5 Bilinear Pairings

For three cyclic groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ of the same prime order p, a *bilinear pairing* e is a map $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$ with the following properties.

- **Bilinearity:** For $g \in \mathbb{G}$, $\hat{g} \in \hat{\mathbb{G}}$ and $a, b \in \mathbb{Z}_p$, $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$.
- **Non-degeneracy:** The map does not send all pairs in $\mathbb{G} \times \hat{\mathbb{G}}$ to unity $\in \mathbb{G}_T$.

If there is an efficient algorithm to compute $e(g, \hat{g})$ for any $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$, the pairing e is called *admissible*. We also expect that it is not feasible to invert a pairing. All pairings considered in our work are admissible and infeasible to invert. Such groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ are

[2]Note that we are using the symbol t instead of the generally used q for the DHI and similar assumptions as the t in t-DHI is directly related to the threshold t in our distributed setting.

called *pairing-friendly* groups. We refer readers to [BSS05, Chap. IX and X] for a detailed mathematical discussion of bilinear pairings.

Following a survey by Galbraith *et al.* [GPS08], there are three types of pairings for prime order groups: namely, type 1, 2, and 3. In *type* 1 pairings, an isomorphism $\varphi : \hat{\mathbb{G}} \to \mathbb{G}$ as well as its inverse φ^{-1} are efficiently computable. These are also called *symmetric pairings* as for such pairings $e(g, \hat{g}) = e(\varphi(\hat{g}), \varphi^{-1}(g))$ for any $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. In *type* 2 pairings, only the isomorphism φ, but not φ^{-1}, is efficiently computable. Finally in *type* 3 pairings, neither of φ nor φ^{-1} can be efficiently computed. The efficiency of the pairing computation improves from type 1 to type 2 to type 3 pairings. For a detailed discussion of the performance aspects of pairings we refer the reader to [GPS08].

We consider the applicability of all three types of pairing while defining our protocols. The major challenge is to accomodate the difficulty of hashing to $\hat{\mathbb{G}}$ and unknown isomorphism φ^{-1}, in type 2 pairings. While in type 3 pairing, the major challenge is to accomodate the inefficiency to compute both φ and φ^{-1}.

In our protocols, we use the bilinear versions of the CDH and t-DHI assumptions.

Definition 2.9. Bilinear Diffie-Hellman (BDH) Assumption. *Given a bilinear group* $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T \rangle$ *where the order of all groups is a common prime* p, *generators* $g \in \mathbb{G}$, $\hat{g} \in \hat{\mathbb{G}}$ *and* $g_T = e(g, \hat{g}) \in \mathbb{G}_T$, *and a tuple* $\langle g^a, \hat{g}^a, g^b, \hat{g}^c \rangle$ *where* $a, b, c \in_R \mathbb{Z}_p$, *for every adversary* $\mathcal{A}_{\mathsf{BDH}}$, $\Pr[\mathcal{A}_{\mathsf{BDH}}(g, \hat{g}, g_T, g^a, \hat{g}^a, g^b, \hat{g}^c) = g_T^{abc}] = \epsilon(\kappa)$.

Definition 2.10. t-Bilinear Diffie-Hellman Inversion (t-BDHI) Assumption. *Let* $\alpha \in_R \mathbb{Z}_p^*$. *Given a bilinear group* $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T \rangle$ *where the order of all groups is a common prime* p, *generators* $g \in \mathbb{G}$, $\hat{g} \in \hat{\mathbb{G}}$ *and* $g_T = e(g, \hat{g}) \in \mathbb{G}_T$, *and a* $(2t+2)$-*tuple* $\langle g, g^\alpha, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^\alpha, \ldots, \hat{g}^{\alpha^t} \rangle \in \mathbb{G}^{t+1} \times \hat{\mathbb{G}}^{t+1}$, *for every PPT algorithm* $\mathcal{A}_{t\text{-}\mathsf{BDHI}}$, *the probability* $\Pr[\mathcal{A}_{t\text{-}\mathsf{BDHI}}(g, g^\alpha, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^\alpha, \ldots, \hat{g}^{\alpha^t}) = {g_T}^{\frac{1}{\alpha}}] = \epsilon(\kappa)$.

We also need the decision version of the BDH assumption, the DBDH assumption.

Definition 2.11. Decision Bilinear Diffie-Hellman (DBDH) Assumption. *Given a bilinear group* $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T \rangle$ *where the order of all groups is a common prime* p, *generators* $g \in \mathbb{G}$, $\hat{g} \in \hat{\mathbb{G}}$ *and* $g_T = e(g, \hat{g}) \in \mathbb{G}_T$, *and a tuple* $\langle g^a, \hat{g}^a, g^b, \hat{g}^c, h_T \rangle$ *where* $h_T \in_R \mathbb{G}_T$ *and* $a, b, c \in_R \mathbb{Z}_p$, *for every adversary* $\mathcal{A}_{\mathsf{DBDH}}$, $\Pr[(\mathcal{A}_{\mathsf{DBDH}}(g, \hat{g}, g_T, g^a, \hat{g}^a, g^b, \hat{g}^c, g_T^{abc}) = \mathit{true}) - (\mathcal{A}_{\mathsf{DBDH}}(g, \hat{g}, g_T, g^a, \hat{g}^a, g^b, \hat{g}^c, h_T) = \mathit{true})] = \epsilon(\kappa)$.

2.6 Homomorphic Commitments

Commitment schemes are fundamental components of many cryptographic protocols. A commitment scheme allows a *committer* to publish a value, called the *commitment* (say

$\mathcal{C}$), which binds her to a message m (*binding*) without revealing it (*hiding*). Later, she may *open* the commitment $\mathcal{C}$ and reveal the committed message m to a verifier, who can check that the message is consistent with the commitment. Damgård surveys the basics of commitment schemes in [Dam99].

Let $\mathcal{C}(\alpha, [r])$ be a commitment to α, where r is an optional randomness parameter. For the homomorphic commitments we use, given $C_1 = \mathcal{C}(\alpha_1, [r_1])$ and $C_2 = \mathcal{C}(\alpha_2, [r_2])$, we have $C_1 \cdot C_2 = \mathcal{C}(\alpha_1 + \alpha_2, [r_1 + r_2])$.

Let $g, h \in_R \mathbb{G}$. The DLog commitment scheme is the most commonly used homomorphic commitment. It is of the form $\mathcal{C}_{\langle g \rangle}(\alpha) = g^\alpha$ with computational hiding (secrecy) under the DLog assumption and unconditional binding (correctness). Pedersen [Ped91b] presented another homomorphic commitment of the form $\mathcal{C}_{\langle g,h \rangle}(\alpha, r) = g^\alpha h^r$ with unconditional hiding but computational binding under the DLog assumption.[3] In the thesis, we extensively use both of these commitment schemes.

2.7 Non-Interactive Zero-Knowledge Proofs

We assume the random oracle model in our work, and so we can use non-interactive zero-knowledge (NIZK) proofs based on the Fiat-Shamir methodology [FS86]. In particular, we use the following three NIZK proofs in our work:

NIZK Proof-of-Knowledge of a Discrete Logarithm. NIZK proof-of-knowledge (NIZKPK) of a discrete logarithm [CS97] is a standard proof-of-knowledge in cryptography. Here, given $\mathcal{C}_{\langle g \rangle}(s) = g^s$ for $g \in \mathbb{G}$ and $s \in \mathbb{Z}_p$, a prover proves that she knows s. We denote the proof as

$$\mathrm{NIZKPK}_{\mathsf{DLog}}(s, \mathcal{C}_{\langle g \rangle}(s)) = \pi_{\mathsf{DLog}} \in \mathbb{Z}_p^2. \tag{2.1}$$

This proof of knowledge is basically a Schnorr signature [Sch91] on message $(g, \mathcal{C}_{\langle g \rangle}(s))$ and is generated as follows:

1. Pick $v \in_R \mathbb{Z}_p$ and let $t = g^v$.
2. Compute hash $c = \mathrm{H}_{\mathsf{DLog}}(g, \mathcal{C}_{\langle g \rangle}(s), t)$, where $\mathrm{H}_{\mathsf{DLog}} : \mathbb{G}^3 \to \mathbb{Z}_p$ is a random oracle hash function.
3. Let $u = v - c \cdot s$.
4. Send the proof $\pi_{\mathsf{DLog}} = (c, u)$ along with $\mathcal{C}_{\langle g \rangle}(s)$.

[3]In an elementary form, Pedersen commitments were introduced by Chaum *et al.* [CDvdG87].

The verifier checks this proof (given π_{DLog}, g, $\mathcal{C}_{\langle g\rangle}(s)$) as follows:

1. Let $t' = g^u(\mathcal{C}_{\langle g\rangle}(s))^c$.
2. Accept the proof as valid if $c = \mathrm{H}_{\mathsf{DLog}}(g, \mathcal{C}_{\langle g\rangle}(s), t')$.

NIZK Proof-of-Equivalence of Commitments. Here, given commitments $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h\rangle}(s,r) = g^s h^r$ to the same value s for generators $g, h \in \mathbb{G}$ and $s, r \in \mathbb{Z}_p$, a prover proves that she knows s and r such that $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h\rangle}(s,r) = g^s h^r$. We denote this by

$$\mathrm{NIZKPK}_{\equiv Com}(s, r, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r)) = \pi_{\equiv Com} \in \mathbb{Z}_p^3. \tag{2.2}$$

The proof is equivalent to zero-knowledge proofs of knowledge used by Canetti *et al.* in their adaptive secure DKG [CGJ$^+$99]. It is generated as follows:

- Pick $v_1, v_2 \in_R \mathbb{Z}_p$, and let $t_1 = g^{v_1}$ and $t_2 = h^{v_2}$.
- Compute hash $c = \mathrm{H}_{\equiv Com}(g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r), t_1, t_2)$, where $\mathrm{H}_{\equiv Com} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash function.
- Let $u_1 = v_1 - c \cdot s$ and $u_2 = v_2 - c \cdot r$.
- Send the proof $\pi_{\equiv Com} = (c, u_1, u_2)$ along with $\mathcal{C}_{\langle g\rangle}(s)$ and $\mathcal{C}_{\langle g,h\rangle}(s,r)$.

The verifier checks this proof (given $\pi_{\equiv Com}$, g, h, $\mathcal{C}_{\langle g\rangle}(s)$, $\mathcal{C}_{\langle g,h\rangle}(s,r)$) as follows:

- Let $t_1' = g^{u_1}\mathcal{C}_{\langle g\rangle}(s)^c$ and $t_2' = h^{u_2}(\frac{\mathcal{C}_{\langle g,h\rangle}(s,r)}{\mathcal{C}_{\langle g\rangle}(s)})^c$.
- Accept the proof as valid if $c = \mathrm{H}_{\equiv Com}(g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r), t_1', t_2')$.

NIZK Proof-of-Equality of Two Discrete Logarithms. Here, given $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle h\rangle}(s) = h^s$, a prover proves equality of the associated discrete logarithms. We denote this proof as

$$\mathrm{NIZKPK}_{\equiv DLog}(s, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle h\rangle}(s)) = \pi_{\equiv DLog} \in \mathbb{Z}_p^2. \tag{2.3}$$

The proof is standard [CP92] and is generated as follows:

- Pick $v \in_R \mathbb{Z}_p$ and let $t_1 = g^v$, $t_2 = h^v$.
- Compute hash $c = \mathrm{H}_{\equiv DLog}(g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle h\rangle}(s), t_1, t_2)$, where $\mathrm{H}_{\equiv DLog} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash function.

- Let $u = v - c \cdot s$.
- Send the proof is $\pi_{\equiv DLog} = (c, u)$ along with $\mathcal{C}_{\langle g \rangle}(s)$ and $\mathcal{C}_{\langle h \rangle}(s)$.

The verifier checks this proof (given $\pi_{\equiv DLog}$, g, h, $\mathcal{C}_{\langle g \rangle}(s)$, $\mathcal{C}_{\langle h \rangle}(s)$) as follows:

- Let $t'_1 = g^u \mathcal{C}_{\langle g \rangle}(s)^c$ and $t'_2 = h^u \mathcal{C}_{\langle h \rangle}(s)^c$.
- Accept the proof as valid if $c = \mathrm{H}_{\equiv DLog}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle h \rangle}(s), t'_1, t'_2)$.

There exists an easier way to prove this equality of DLogs if a pairing between the groups generated by g and h is available. Using a technique due to Joux and Nguyen [JN03] to solve the DDH problem over pairing-friendly groups, given g^x and $h^{x'}$ the verifier checks if $e(g, h^{x'}) \overset{?}{=} e(g^x, h)$ to see whether $x \overset{?}{=} x'$. However, when using a type 3 pairing, in the absence of an efficient isomorphism between $\mathbb{G}$ and $\hat{\mathbb{G}}$, if both g and h belong to the same group then the pairing-based scheme does not work. It also does not work for a type 2 pairing if $g, h \in \mathbb{G}$. In these situations, $\mathrm{NIZKPK}_{\equiv DLog}$ provides a less efficient but completely practical alternative there.

Part I

Distributed Key Generation

Chapter 3

Polynomial Commitments and Their Applications to Secret Sharing

3.1 Preliminaries

A verification mechanism for a consistent dealing is fundamental to VSS. It is achieved using distributed computing techniques in the information-theoretical setting, but in the computational security setting, where Shamir's secret sharing methodology is prominent, it is achieved by committing to a polynomial $\phi(x) \in \mathbb{Z}_p[x]$.

Suppose ϕ has degree t and coefficients $\phi_0, \ldots, \phi_t$. The most obvious way to commit to $\phi(x)$ is to commit to the string $(\phi_0||\phi_1||\ldots||\phi_t)$, or to some other unambiguous string representation of ϕ. Based on the commitment function used, this option may have a constant-size commitment that uniquely determines ϕ. However, it limits the options for opening the commitment; an opening reveals the entire polynomial. This is not always suitable for cryptographic applications, most notably VSS, that require evaluations $\phi(i)$ for indices $i \in \mathbb{Z}_p$ of polynomial $\phi(x)$ be revealed to different parties or at different points in the protocol.

In the VSS literature, this is achieved using *homomorphic commitments*. VSS schemes in the literature utilize the DLog and Pedersen commitments that we discused in Section 2.6. For example, one commits to the coefficients of $\phi(x)$ as $\vec{\mathcal{C}} = (g^{\phi_0}, \ldots, g^{\phi_t})$, where each element is a DLog commitment to a coefficient. This allows anybody to easily confirm that an opening $\phi(i)$ for index i is consistent with commitment $\vec{\mathcal{C}}$. However, the size of a commitment to polynomial $\phi(x)$ of degree t is not constant; it is $t + 1 = \omega(1)$.

Contributions. Our main contribution here is an efficient scheme to commit to polynomials $\phi(x) \in \mathbb{Z}_p[x]$, called $\mathsf{PolyCommit}$ ($\mathsf{PolyCommit}_{DL}$ in [KZG10]), with the following main features:

- The size of the commitment is constant, a single group element.
- The committer can efficiently open the commitment to any correct evaluation $\phi(i)$ along with an element called the *witness*, which allows a verifier to confirm that $\phi(i)$ is indeed the evaluation at i of the polynomial $\phi(x)$.

The construction is based on an algebraic property of polynomials $\phi(x) \in \mathbb{Z}_p[x]$ that $(x-i)$ *perfectly* divides the polynomial $\phi(x) - \phi(i)$ for any $i \in \mathbb{Z}_p$. When a set of evaluations $\{\phi(s) : s \in S\}$ is opened at the same time, what we term *batch opening*, the overhead still remains a *single* witness element. The hiding property of the scheme is based on the DLog assumption. The binding property of the main scheme is proven under the t-strong Diffie-Hellman (t-SDH) assumption [BB04b], while the security of batch opening assumes that the bilinear version of the t-SDH problem [Goy07] is hard. Further, our scheme $\mathsf{PolyCommit}$ is homomorphic and easily randomizable.

We use $\mathsf{PolyCommit}$ to define a computational VSS protocol in the synchronous communication model. The new VSS protocol requires a broadcast with size $O(1)$ as compared to $O(n)$ required in the best known protocols (where n is the number of nodes). We then transform the above the VSS protocol into a synchronous DKG protocol.

In the rest of section, we cover some preliminary related work. In Section 3.2, we describe our cryptographic assumptions. Section 3.3 defines polynomial commitments, and presents and prove our construction. In Section 3.4, we describe our synchronous VSS scheme, while in Section 3.5, we convert it to a DKG protocol.

Related Work. Similar to our scheme, a Merkle hash tree [MOV97] allows many values to be committed to with a single element. Here, the leaves of a binary tree are the messages. Each non-leaf node has the value $H(L||R)$, where L and R are its children and H is a collision-resistant hash function. One can open the commitment to an individual message by revealing the message, and a path up the tree to the root. The opening has size $O(\log n)$ as compared to $O(1)$ in our scheme. Further, this scheme is not homomorphic.

Chase *et al.* [CHL$^+$05] introduce mercurial commitments to construct zero-knowledge sets, which eventually led to commitment schemes for committing to a vector of messages [CFM08, LY10]. Catalano *et al.* [CFM08], and Libert & Yung [LY10] construct vector commitment schemes under the name *trapdoor t-mercurial commitments*. The security of both the commitment schemes is based on "SDH-like" assumptions and their system parameters have size $O(t)$, as in our scheme. In [CFM08], all messages must be revealed when opening, while in [LY10], the committer may open a commitment to a single message. However, for commitments in [LY10], as each of the t committed messages is associated with an element in the system parameters g^{α^i} for $i = 1, \ldots, t$, they are indexed from 1 to t. It is not possible to have arbitrary indices for the committed values and generate a commitment for a set of (key/index, value) pairs. Our scheme does not have any such restriction on the domain for the indices and it can be used to generate a commitment for any set of (key, value) pairs in $\mathbb{Z}_p \times \mathbb{Z}_p$.

Another related primitive is a cryptographic *accumulator* [BdM93], which aggregates a large set of input elements into a single element and can provide a witness as evidence that an element is included in the accumulator. Further, it is possible to use a witness to prove (in zero-knowledge) that the element is included in the accumulator. Camenisch and Lysyanskaya [CL02] extend the concept to *dynamic accumulators*, which support efficient updates. Au *et al.* [AWSM07] observe that a paring-based accumulator by Nguyen [Ngu05] is a *bounded* accumulator; that is, only a fixed number of elements can be accumulated. They then go on to use bounded accumulators to construct a compact e-cash scheme [ASM08]. However, the accumulated elements in this scheme are not ordered, which makes it inappropriate for accumulating polynomials. While $\mathsf{PolyCommit}$ provides the same features as non-dynamic accumulators, it has additional features (hiding, batch opening, homomorphism) and is more general since we can commit to a polynomial instead of a set.

3.2 Assumptions

In this section, we describe the cryptographic primitives and the number-theoretic assumptions that we utilize in this chapter.

We use the concepts of bilinear pairing and NIZKPK of a discrete logarithm. For three cyclic multiplicative pairing groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ of the same prime order p, e is a bilinear pairing map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. For $\mathcal{C}_{\langle g \rangle}(s) = g^s$ for $g \in \mathbb{G}$ and $s \in \mathbb{Z}_p$, NIZKPK of a discrete logarithm is described as $\text{NIZKPK}_{\mathsf{DLog}}(s, \mathcal{C}_{\langle g \rangle}(s)) = \pi_{\mathsf{DLog}} \in \mathbb{Z}_p^2$. The mathematical details of these concepts are already discussed in Section 2.5 and Section 2.7 respectively.

Further, we use the DLog assumption described in Section 2.4, and the *t-strong Diffie-Hellman* (t-SDH) assumption [BB04b] to prove the security of the basic $\mathsf{PolyCommit}$ scheme. The security of two additional properties of the scheme requires a generalization of the *t-Diffie-Hellman inversion* (t-DHI) assumption defined in Section 2.4, and the bilinear version of t-SDH, the t-BSDH assumption [Goy07].

The t-DHI problem is, on input $\langle g, g^{\alpha}, \ldots, g^{\alpha^t} \rangle \in \mathbb{G}^{t+1}$ to output $g^{1/\alpha}$, or equivalently (see [BBG05]), $g^{\alpha^{t+1}}$. In this work, we use a generalization of the t-DHI assumption, where $\mathcal{A}$ has to output a pair $\langle \phi(x), g^{\phi(\alpha)} \rangle \in \mathbb{Z}_p[x] \times \mathbb{G}$ such that $2^{\kappa} \gg \deg(\phi) > t$. We call this assumption the *t-polynomial Diffie-Hellman* (t-polyDH) assumption. This assumption was implicitly made by [AWSM07, ASM08] to support their claim that the accumulator of [Ngu05] is bounded.

Definition 3.1. **t-Polynomial Diffie-Hellman (t-polyDH) Assumption.** *Let $\alpha \in_R \mathbb{Z}_p^*$. Given as input a $(t+3)$-tuple $\langle g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}\rangle \in \mathbb{G}^{t+1} \times \hat{\mathbb{G}}^2$, for every adversary $\mathcal{A}_{t\text{-polyDH}}$, the probability $\Pr[\mathcal{A}_{t\text{-polyDH}}(g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}) = \langle \phi(x), g^{\phi(\alpha)}\rangle] = \epsilon(\kappa)$ for any freely chosen $\phi(x) \in \mathbb{Z}_p[x]$ such that $2^{\kappa} \gg \deg(\phi) > t$.*

Boneh and Boyen [BB04b] defined the t-SDH assumption that is related to but stronger than the t-DHI assumption and has exponentially many non-trivially different solutions, all of which are acceptable.

Definition 3.2. **t-Strong Diffie-Hellman (t-SDH) Assumption.** *Let $\alpha \in_R \mathbb{Z}_p^*$. Given as input a $(t+3)$-tuple $\langle g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}\rangle \in \mathbb{G}^{t+1} \times \hat{\mathbb{G}}^2$, for every adversary $\mathcal{A}_{t\text{-SDH}}$, the probability $\Pr[\mathcal{A}_{t\text{-SDH}}(g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}) = \langle c, g^{\frac{1}{\alpha+c}}\rangle] = \epsilon(\kappa)$ for any value of $c \in \mathbb{Z}_p^* \backslash \{-\alpha\}$.*

Security of the batch opening extension of our PolyCommit scheme requires the bilinear version of the t-SDH assumption, the t-BSDH assumption [Goy07]. Here, we consider the asymmetric pairing version of this assumption.

Definition 3.3. **t-Bilinear Strong Diffie-Hellman (t-BSDH) Assumption.** *Let $\alpha \in_R \mathbb{Z}_p^*$. Given as input a $(2t+2)$-tuple $\langle g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}, \ldots, \hat{g}^{\alpha^t}\rangle \in \mathbb{G}^{t+1} \times \hat{\mathbb{G}}^{t+1}$, for every adversary $\mathcal{A}_{t\text{-BSDH}}$, the probability $\Pr[\mathcal{A}_{t\text{-BSDH}}(g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}, \ldots, \hat{g}^{\alpha^t}) = \langle c, e(g, \hat{g})^{\frac{1}{\alpha+c}}\rangle] = \epsilon(\kappa)$ for any value of $c \in \mathbb{Z}_p^* \backslash \{-\alpha\}$.*

A similar assumption for symmetric pairings was made in [GMC07], but with a different solution: $\langle c, e(g, h)^{1/(\alpha+c)}\rangle$, where $h \in_R \mathbb{G}$ is an additional system parameter.

3.3 Constant-Size Polynomial Commitment

In this section, we provide a formal definition of a polynomial commitment scheme, followed by an efficient construction based on the DLog and t-SDH assumptions. We also prove the security properties and discuss some special useful features of our construction.

3.3.1 Definition

A polynomial commitment scheme consists of six algorithms: Setup, Commit, Open, VerifyPoly, CreateWitness, and VerifyEval.

Setup is run by a trusted or distributed authority. Setup performs the basic setup for the commitment scheme and generates a secret key SK and a public key tuple PK. Notably, the trapdoor SK is not required in the rest of the protocol, but may be used by a simulator to "forge" openings consistent with any commitment. Any party P may use Commit to commit to a polynomial $\phi(x)$. Denote this commitment $\mathcal{C}$. Open simply opens the commitment by revealing $\phi(x)$, which can be verified by $\mathsf{VerifyPoly}$. Most of the cryptographic applications of a polynomial in $\mathbb{Z}_p[x]$ use its evaluations $\phi(i)$ for some $i \in \mathbb{Z}_p$. $\mathsf{CreateWitness}$ provides a witness w_i for the evaluation $\phi(i)$ of the committed polynomial for an index i. The receiver uses $\mathsf{VerifyEval}$ and w_i to verify if the index-value pair $\langle i, \phi(i)\rangle$ is consistent with $\mathcal{C}$.

Importantly, the size of commitments and individual witnesses should be independent of the degree of the committed polynomial.

Setup (1^κ) generates an appropriate algebraic structure $\mathcal{G}$ and a commitment public-private key pair $\langle \mathsf{PK}, \mathsf{SK}\rangle$. For simplicity, we add $\mathcal{G}$ to the public key PK. Setup is run by a trusted or distributed authority. Note that SK is not required in the rest of the scheme.

Commit $(\mathbf{PK}, \phi(x))$ outputs a commitment $\mathcal{C}$ to a polynomial $\phi(x)$ for the public key PK.

Open $(\mathbf{PK}, \mathcal{C}, \phi(x))$ outputs the polynomial $\phi(x)$ used while creating the commitment.

$\mathsf{VerifyPoly}$ $(\mathbf{PK}, \mathcal{C}, \phi(x))$ verifies that $\mathcal{C}$ is a commitment to $\phi(x)$. If so the algorithm outputs 1, otherwise it outputs 0.

$\mathsf{CreateWitness}$ $(\mathbf{PK}, \phi(x), i)$ outputs $\langle i, \phi(i), w_i\rangle$, where w_i is a witness for the evaluation $\phi(i)$ of $\phi(x)$ at the index i.

$\mathsf{VerifyEval}$ $(\mathbf{PK}, \mathcal{C}, i, \phi(i), w_i)$ using w_i as a witness, verifies that $\phi(i)$ is indeed the evaluation at the index i of the polynomial committed in $\mathcal{C}$. If so the algorithm outputs 1, otherwise it outputs 0.

Informally, in terms of security, a malicious committer should not be able to convincingly present two different values as $\phi(i)$ with respect to $\mathcal{C}$. Further, until more than $\deg(\phi)$ points are revealed, the adversary should not be able to compute the polynomial $\phi(x)$. Next, we formally define security and correctness of a polynomial commitment.

Definition 3.4. *Let $\langle PK, SK\rangle \leftarrow Setup(1^\kappa)$ and $\mathcal{C} \leftarrow Commit(PK, \phi(x))$. We say (Setup, Commit, Open, VerifyPoly, CreateWitness, and VerifyEval) is a* secure polynomial commitment scheme *if it satisfies the following properties.*

Correctness. *For all $\phi(x) \in \mathbb{Z}_p[x]$ and all commitments $\mathcal{C}$ output by $Commit(PK, \phi(x))$*

- *the output of* $\mathsf{Open}(\mathsf{PK}, \mathcal{C}, \phi(x))$ *is successfully verified by* $\mathsf{VerifyPoly}(\mathsf{PK}, \mathcal{C}, \phi(x))$ *and*
- *any* $\langle i, \phi(i), w_i\rangle$ *output by* $\mathsf{CreateWitness}(\mathsf{PK}, \phi(x), i)$ *is successfully verified by* $\mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, \phi(i), w_i)$.

Polynomial Binding. *For all adversaries* $\mathcal{A}$*:*

$$\Pr\begin{pmatrix} \mathsf{PK} \leftarrow \mathsf{Setup}(1^\kappa),\ \langle \mathcal{C}, \phi(x), \phi'(x)\rangle \leftarrow \mathcal{A}(\mathsf{PK}) : \\ \mathsf{VerifyPoly}(\mathsf{PK}, \mathcal{C}, \phi(x)) = 1\ \wedge \\ \mathsf{VerifyPoly}(\mathsf{PK}, \mathcal{C}, \phi'(x)) = 1\ \wedge \\ \phi(x) \neq \phi'(x) \end{pmatrix} = \epsilon(\kappa).$$

Evaluation Binding. *For all adversaries* $\mathcal{A}$*:*

$$\Pr\begin{pmatrix} \mathsf{PK} \leftarrow \mathsf{Setup}(1^\kappa),\ (\langle \mathcal{C}, i, \phi(i), w_i, \phi'(i), w_i'\rangle) \leftarrow \mathcal{A}(\mathsf{PK}) : \\ \mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, \phi(i), w_i) = 1\ \wedge \\ \mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, \phi'(i), w_i') = 1\ \wedge \\ \phi(i) \neq \phi'(i) \end{pmatrix} = \epsilon(\kappa).$$

Hiding. *Given* $\langle \mathsf{PK}, \mathcal{C}\rangle$ *and* $\{\langle i_j, \phi(i_j), w_{\phi_{i_j}}\rangle\}_{j=1}^{\deg(\phi)}$, *no adversary* $\mathcal{A}$ *can determine* ϕ *with non-negligible probability.*

3.3.2 Construction: PolyCommit

We now provide an efficient construction of a polynomial commitment scheme, as defined in Section 3.3.1. PolyCommit is based on an algebraic property of polynomials $\phi(x) \in \mathbb{Z}_p[x]$: $(x - i)$ perfectly divides the polynomial $\phi(x) - \phi(i)$ for $i \in \mathbb{Z}_p$. In the literature, Herzberg *et al.* [HJKY95] have used this technique in their proactive share recovery scheme.

Our construction uses a parameter t, which is not present in the general definition. This is a bound on the degree of polynomials that may be committed. In Section 3.3.4, we show that it is not possible to commit to a polynomial of degree higher than t provided the t-polyDH assumption holds.

Setup$(1^\kappa, t)$ computes three groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ of prime order p (providing κ-bit security) such that there exists a bilinear pairing $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$ and for which the t-SDH assumption holds. We denote the generated bilinear pairing group as $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T\rangle$. Choose random generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. Let $\alpha \in_R \mathbb{Z}_p^*$ be SK, generated by a (possibly distributed) trusted authority. Setup also generates a $(t+3)$-tuple $\langle g, g^\alpha, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^\alpha\rangle \in \mathbb{G}^{t+1} \times \hat{\mathbb{G}}^2$ and outputs $\mathsf{PK} = \langle \mathcal{G}, g, g^\alpha, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^\alpha\rangle$ as output. Note that SK is not required by the other algorithms of the commitment scheme, and it can be discarded by the authority if t is fixed.

Commit$(\mathsf{PK}, \phi(x))$ computes the commitment $\mathcal{C} = g^{\phi(\alpha)} \in \mathbb{G}$ for polynomial $\phi(x) \in \mathbb{Z}_p[X]$ of degree t or less. For $\phi(x) = \sum_{j=0}^{\deg(\phi)} \phi_j x^j$, it outputs $\mathcal{C} = \prod_{j=0}^{\deg(\phi)} (g^{\alpha^j})^{\phi_j}$ as the commitment to $\phi(x)$.

Open$(\mathsf{PK}, \mathcal{C}, \phi(x))$ outputs the committed polynomial $\phi(x)$.

VerifyPoly$(\mathsf{PK}, \mathcal{C}, \phi(x))$ verifies that $\mathcal{C} \overset{?}{=} g^{\phi(\alpha)}$. If $\mathcal{C} = \prod_{j=0}^{\deg(\phi)} (g^{\alpha^j})^{\phi_j}$ the algorithm outputs 1, else it outputs 0. Note that this only works when $\deg(\phi) \leq t$.

CreateWitness$(\mathsf{PK}, \phi(x), i)$ computes $\psi_i(x) = \frac{\phi(x)-\phi(i)}{(x-i)} \in \mathbb{Z}_p[x]$ and outputs $\langle i, \phi(i), w_i \rangle$, where the witness $w_i = g^{\psi_i(\alpha)}$. As mentioned above, $(x - i)$ perfectly divides $\phi(x) - \phi(i)$.

VerifyEval$(\mathsf{PK}, \mathcal{C}, i, \phi(i), w_i)$ verifies that $\phi(i)$ is the evaluation at the index i of the polynomial committed to by $\mathcal{C}$. If $e(\mathcal{C}, \hat{g}) \overset{?}{=} e(w_i, \hat{g}^\alpha/\hat{g}^i)e(g, \hat{g})^{\phi(i)}$, the algorithm outputs 1, else it outputs 0.

3.3.3 Analysis

Theorem 3.5. *PolyCommit is a secure polynomial commitment scheme (as defined in Definition 3.4) provided the* DLog *and* t-SDH *assumptions hold in* $\mathcal{G}$.

Proof. We prove that the all properties of Definition 3.4 hold.

Correctness. Correctness of VerifyPoly is obvious. VerifyEval is correct because

$$\begin{aligned} e(w_i, \hat{g}^\alpha/\hat{g}^i)e(g, \hat{g})^{\phi(i)} &= e(g^{\psi_i(\alpha)}, \hat{g}^{(\alpha-i)})e(g, \hat{g})^{\phi(i)} \\ &= e(g, \hat{g})^{\psi_i(\alpha)(\alpha-i)+\phi(i)} \\ &= e(g, \hat{g})^{\phi(\alpha)} \text{ as } \phi(x) = \psi_i(x)(x-i) + \phi(i). \end{aligned}$$

Polynomial Binding. Suppose there exists an adversary $\mathcal{A}$ that breaks polynomial binding for a commitment $\mathcal{C}$ by outputting two polynomials $\phi(x)$ and $\phi'(x)$ that are accepted by VerifyPoly. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to efficiently compute $SK = \alpha$.

For $\phi(x)$ and $\phi'(x)$ generated by $\mathcal{A}$, $\mathcal{C} = g^{\phi(\alpha)} = g^{\phi'(\alpha)}$. For a polynomial $\phi''(x) = \phi(x) - \phi'(x) \in \mathbb{Z}_p[x]$, the corresponding $C_{\phi''} = g^{\phi''(\alpha)} = g^{\phi(\alpha)}/g^{\phi'(\alpha)} = 1$ as the commitment scheme is homomorphic in nature. Therefore, $\phi''(\alpha) = 0$, i.e., α is a root of polynomial $\phi''(x)$ and by factoring $\phi''(x)$ [Sho06, Chap. 21], $\mathcal{B}$ can easily find $\mathsf{SK} = \alpha$.[1] After computing α, $\mathcal{B}$ can solve the instance of the t-SDH problem given by the system parameters.

[1]There may be up to t choices here; but the correct one is easy to verify using PK.

Evaluation Binding. Suppose there exists an adversary $\mathcal{A}$ that breaks the evaluation binding property of commitment $\mathcal{C}$ and computes two witness tuples $\langle i, y_i, w_i\rangle$ and $\langle i, y_i', w_i'\rangle$ for index i that are accepted by $\mathsf{VerifyEval}$. We show how to construct an algorithm $\mathcal{B}$, using $\mathcal{A}$, that can break the t-SDH assumption.

$\mathcal{B}$ presents the t-SDH instance $\langle \mathcal{G}, g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}\rangle$ as PK to $\mathcal{A}$. $\mathcal{A}$ outputs a commitment $\mathcal{C}$, and two witness tuples $\langle i, y_i, w_i\rangle$ and $\langle i, y_i', w_i'\rangle$ for an index $i \in \mathbb{Z}_p$ such that $e(\mathcal{C}, \hat{g}) = e(w_i, \hat{g}^{\alpha-i})e(g, \hat{g})^{y_i} = e(w_i', \hat{g}^{\alpha-i})e(g, \hat{g})^{y_i'}$ and $y_i \neq y_i'$. Note that $y_i \neq y_i' \Rightarrow i \neq \alpha$. For $v = \log_g w_i$ and $v' = \log_g w_i'$, we have

$$\begin{aligned} v(\alpha - i) + y_i &= v'(\alpha - i) + y_i' \\ \frac{v - v'}{y_i' - y_i} &= \frac{1}{\alpha - i} \,. \end{aligned}$$

Therefore, algorithm $\mathcal{B}$ computes

$$\begin{aligned} \left(\frac{w_i}{w_i'}\right)^{\frac{1}{y_i' - y_i}} &= g^{\frac{v - v'}{y_i' - y_i}} \\ &= g^{\frac{1}{\alpha - i}} \end{aligned}$$

and returns $\langle -i, g^{\frac{1}{\alpha-i}}\rangle$ as a solution for the t-SDH instance. It is easy to see that the success probability of solving the instance is the same the success probability of $\mathcal{A}$, and the time required is a small constant larger than the time required by $\mathcal{A}$.

Hiding. Suppose there exists an adversary $\mathcal{A}$ that breaks the hiding property of commitment $\mathcal{C}$ and correctly computes polynomial $\phi(x)$ (without loss of generality $\deg \phi = t$) given t valid witness tuples $\langle i, y_i, w_i\rangle$. We show how to use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ than can break the DLog assumption.

Let $\langle g, g^a\rangle \in \mathbb{G}^2$ be a DLog instance that $\mathcal{B}$ needs to solve. $\mathcal{B}$ generates PK for $\mathcal{A}$ by randomly picking $\alpha \in_R \mathbb{Z}_p^*$ and computing $\mathsf{PK} = \langle \mathcal{G}, g, g^{\alpha}, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^{\alpha}\rangle$. $\mathcal{B}$ sets $\langle j, y_j\rangle \in_R \mathbb{Z}_p^2$ as polynomial $\phi(x)$'s evaluations at indices j. It then assumes $\phi(0) = a$, which is the answer for the DLog instance and computes $g^{\phi(\alpha)}$ using $t+1$ exponentiated evaluations: $\langle 0, g^a\rangle$ and t other chosen pairs $\langle j, g^{y_j}\rangle$. Finally, $\mathcal{B}$ computes witnesses w_j for t chosen evaluations $\langle j, y_j\rangle$ as $w_j = (g^{\phi(\alpha)}/g^{y_j})^{\frac{1}{\alpha-j}}$, and sends PK and t witness tuples $\langle j, y_j, w_j\rangle$ to $\mathcal{A}$. Once $\mathcal{A}$ returns polynomial $\phi(x)$, $\mathcal{B}$ returns the constant term $\phi(0)$ as the solution for the DLog instance.

It is easy to see that the success probability of solving the DLog instance is the same the success probability of $\mathcal{A}$, and the time required is a small constant larger than the time required by $\mathcal{A}$. □

Note that the above hiding property is computational and consequently weak. In [KZG10], we have recently extended our results to define a scheme ($\mathsf{PolyCommit}_{Ped}$) with the unconditional hiding property.

3.3.4 Features

Now, we discuss some important non-fundamental features of $\mathsf{PolyCommit}$.

Homomorphism. The $\mathsf{PolyCommit}$ scheme is homomorphic in nature. Observe that given commitments $\mathcal{C}_{\phi_1}$ and $\mathcal{C}_{\phi_2}$ associated with polynomials $\phi_1(x)$ and $\phi_2(x)$ respectively, one can compute the commitment $\mathcal{C}_\phi$ for $\phi(x) = \phi_1(x) + \phi_2(x)$ as $\mathcal{C}_\phi = \mathcal{C}_{\phi_1}\mathcal{C}_{\phi_2}$. Further, given two witness-tuples $\langle i, \phi_1(i), w_{\phi_1 i}\rangle$ and $\langle i, \phi_2(i), w_{\phi_2 i}\rangle$ at index i associated with polynomials $\phi_1(x)$ and $\phi_2(x)$ respectively, the corresponding tuple for index i for polynomial $\phi(x) = \phi_1(x) + \phi_2(x)$ can be given as $\langle i, \phi_1(i) + \phi_2(i), w_{\phi_1 i}w_{\phi_2 i}\rangle$.

Unconditional Hiding. When $k < \deg(\phi)$ evaluations have been revealed, $\mathsf{PolyCommit}$ unconditionally hides any unrevealed evaluation $\phi(i_{k'})$ for an index $i_{k'} \in_R \mathbb{Z}_p$, since the $k+1$ points $\langle\alpha, \phi(\alpha)\rangle, \langle i_1, \phi(i_1)\rangle, \ldots, \langle i_k, \phi(i_k)\rangle$, are insufficient to recover $\phi(i_{k'})$.

Trapdoor Commitment. This construction is also a trapdoor commitment scheme, where $\mathsf{SK} = \alpha$ is the trapdoor. Given α, a *simulator* can create witnesses for arbitrary values with respect to $\mathcal{C} = g^d$ for an unknown d. For example, to "prove" $\phi(i) = r$ (where ϕ is the polynomial supposedly committed to by $\mathcal{C}$), output $w = [\mathcal{C} \cdot g^{-r}]^{1/(\alpha - i)}$. It can easily be checked that $\mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, r, w) = 1$.

Revealing Exponentiated Evaluations. Some applications require a polynomial evaluation $\phi(i)$ to be opened only in an exponentiated form $g^{\phi(i)}$. In that case, the correctness, hiding and polynomial binding properties of $\mathsf{PolyCommit}$ remain intact; however, the evaluation binding property is no longer valid. For a commitment $\mathcal{C} = g^{\phi(\alpha)}$ for polynomial $\phi(x)$, let $\langle i, g^{\phi(i)}, w_i\rangle$ be the correct opening for index i. The adversary can present wrong openings of the form $\langle i, g^{\phi(i) - r(\alpha - i)}, w_i g^r\rangle$ for $r \in_R \mathbb{Z}_p$, which will be accepted by $\mathsf{VerifyEval}$.

Notably, in any of the above wrong openings, the committer does not know the discrete logarithm of the exponentiated evaluations $g^{\phi(i) - r(\alpha - i)}$. Therefore, to avoid the above attack we utilize $\mathrm{NIZKPK}_{\mathsf{DLog}}$ described in Section 2.7. The modified $\mathsf{CreateWitness}$ ($\mathsf{CreateWitness}$') returns $\langle i, g^{\phi(i)}, w_i, \pi_{\mathsf{DLog}} = \mathrm{NIZKPK}_{\mathsf{DLog}}(\phi(i), g^{\phi(i)})\rangle$. Correspondingly, the modified $\mathsf{VerifyEval}$ ($\mathsf{VerifyEval}'(\mathsf{PK}, \mathcal{C}, i, g^{\phi(i)}, w_i, \pi_{\mathsf{DLog}})$) checks π_{DLog} along with $e(\mathcal{C}, \hat{g}) \stackrel{?}{=} e(w_i, \hat{g}^\alpha/\hat{g}^i)e(g^{\phi(i)}, \hat{g})$. Similar to the binding proof of the original $\mathsf{PolyCommit}$ scheme, an adversary that can generate two valid tuples $\langle i, g^{\phi(i)}, w_i, \mathrm{NIZKPK}_{\mathsf{DLog}}(\phi(i), g^{\phi(i)})\rangle$ and $\langle i, g^{\phi'(i)}, w'_i, \mathrm{NIZKPK}_{\mathsf{DLog}}(\phi'(i), g^{\phi'(i)})\rangle$ can be used to break the t-SDH assumption to compute $g^{\frac{1}{\alpha - i}}$. Note that the challenger has to extract $\phi(i)$ and $\phi'(i)$ from the adversary in the proof; this can be achieved using a standard rewinding technique [ADW09, Sec. 4].

Strong Correctness. As discussed in Section 2.2, VSS may require an additional property of PolyCommit, namely, that $t+1$ correct shares (evaluations) are sufficient to reconstruct the polynomial. In other words, it is not possible to commit to a polynomial of degree greater than t. This is referred as the *strong correctness* property of VSS.

Defining strong correctness for PolyCommit is problematic, since for $z \in_R \mathbb{Z}_p$, there are many polynomials ϕ of degree greater than t such that $\phi(\alpha) = z$, and so g^z is trivially a commitment to some polynomial of degree $t' > t$. To avoid this triviality, we require that an adversary $\mathcal{A}$ must convince a challenger $\mathcal{B}$ that he knows ϕ with the following game. $\mathcal{A}$ creates a commitment to a claimed polynomial ϕ of degree t'. $\mathcal{B}$ challenges $\mathcal{A}$ with $t'+1$ indices $I \subset \mathbb{Z}_p$. $\mathcal{A}$ wins if he is able to produce $\langle i, \phi(i), w_i \rangle_{i \in I}$ accepted by VerifyEval such that the interpolation (in exponents) of any t' and $t'+1$ witnesses do not differ. The second condition ensures that $\deg(\phi) = t'$ by ensuring that the degree of the polynomial whose exponentiated evaluations are witnesses is $t'-1$.

Theorem 3.6. *PolyCommit has the strong correctness property if the t-polyDH assumption holds in $\mathcal{G}$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that verifiably commits to a polynomial of degree greater than t using $\mathsf{PK} = \langle \mathcal{G}, g, g^{\alpha}, \ldots, g^{\alpha^t} \rangle$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the t-polyDH assumption.

$\mathcal{B}$ presents the t-polyDH instance $\langle \mathcal{G}, g, g^{\alpha}, \ldots, g^{\alpha^t} \rangle$ as PK to $\mathcal{A}$. $\mathcal{A}$ outputs a commitment $\mathcal{C}$ and t', where t' is the claimed degree of the committed polynomial. If $t' \leq t$, then $\mathcal{B}$ returns failure and stops, else $\mathcal{B}$ chooses and sends a set I of $t'+1$ random indices to $\mathcal{A}$. $\mathcal{A}$ returns $t'+1$ evaluation tuples of the form $\langle i, y_i, w_i \rangle$ for $i \in I$. $\mathcal{B}$ verifies these $t'+1$ evaluation tuples using VerifyEval. If a verification fails, then $\mathcal{B}$ returns failure and stops. Once all verifications are successful, $\mathcal{B}$ interpolates the $t'+1$ values y_i to compute the claimed committed polynomial $\phi'(x)$. If $\deg(\phi') < t'$, then $\mathcal{B}$ returns failure and stops, else $\mathcal{B}$ is assured that $\deg(\phi') = t'$ and therefore $\deg(\phi) \geq t'$. $t'+1$ verifications do not confirm that $\phi'(x)$ is the committed polynomial $\phi(x)$ as $\phi(x)$ can be of degree $> t'$. The verifications only assure that the evaluations of $\phi'(x)$ at the verified indices are equal to those of $\phi(x)$. We can represent $\psi_i(x)|_{x=\alpha} = \psi_i(\alpha) = \frac{\phi(\alpha)-\phi(i)}{\alpha-i} = \frac{\phi(i)-\phi(\alpha)}{i-\alpha} = \psi_\alpha(i) = \psi_\alpha(x)|_{x=i}$. To confirm that the committed polynomial is of degree t' and consequently $\phi(x) = \phi'(x)$, $\mathcal{B}$ interpolates (in exponent) the $t'+1$ w_i values from the evaluation set to generate exponentiated coefficients of the polynomial $\psi_\alpha(x)$. If $\deg(\phi) > t'$, then $\psi_\alpha(x)$ should be of degree $> t'-1$. Therefore, if the coefficient $\psi_{t'}$ associated with $g^{\alpha^{t'}}$ is non-zero, then $\mathcal{B}$ returns failure and stops, else $\mathcal{B}$ returns $\phi(x)$ and $\mathcal{C}$ as an answer to the t-polyDH instance. Although it is possible that $\psi_\alpha(x)$ is of degree $t'-1$ for $\phi(x)$ with degree greater than t', the success probability of $\mathcal{A}$ to achieve that is negligible for the indices randomly chosen by $\mathcal{B}$.

It is easy to see that the success probability of solving the instance is negligibly less than the success probability of $\mathcal{A}$ in verifiably committing to a polynomial of degree greater than t, and the time required is a small constant larger than the time required by $\mathcal{A}$. $\square$

Batch Opening. In the case when multiple evaluations in a commitment must be opened, the opening may be batched to reduce the computation and the communication of both the committer and the verifier. That is, opening multiple evaluations at the same time is cheaper than opening each of those evaluations individually using $\mathsf{CreateWitness}$ and $\mathsf{VerifyEval}$. Let $B \subset \mathbb{Z}_p^*$, $|B| < t$ be a set of indices to be opened, for a commitment $\mathcal{C} = g^{\phi(\alpha)}$ created using $\mathsf{PolyCommit}$. The witness for the values $\phi(i)$, for all $i \in B$, is computed as $w_B = g^{\psi_B(\alpha)}$ for the polynomial $\psi_B(x) = \frac{\phi(x)-r(x)}{\prod_{i\in B}(x-i)}$ where $r(x)$ is the remainder of $\phi(x)/(\prod_{i\in B}(x-i))$. That is, $\phi(x) = \psi_B(x)\left(\prod_{i\in B}(x-i)\right) + r(x)$ and for $i \in B$, $\phi(i) = r(i)$. We define two algorithms for batch opening. The algorithm $\mathsf{CreateWitnessBatch}(\mathsf{PK}, \phi(x), B)$ outputs $\langle B, r(x), w_B\rangle$ and the algorithm $\mathsf{VerifyEvalBatch}$ $(\mathsf{PK}, \mathcal{C}, B, r(x), w_B)$ outputs 1 if $e(\mathcal{C}, \hat{g}) \overset{?}{=} e(w_B, \hat{g}^{\prod_{i\in B}(\alpha-i)})e(g, \hat{g}^{r(\alpha)})$ holds. We observe that the setup has to generate $\langle \hat{g}^{\alpha^2}, \ldots, \hat{g}^{\alpha^{t-1}}\rangle$ along with the usual PK for the verification to work.

In terms of security, since commitments are formed in the same way as the Commit algorithm and $\mathsf{CreateWitnessBatch}$ reveals no more information than running $\mathsf{CreateWitness}$ for all batch elements individually, the hiding property (proven in Theorem 3.5) still holds. For binding, an adversary should not be able to open a batch B containing an index i, in a manner that conflicts with the value $\phi(i)$ output in an individual opening of index i. Formally, we say that batch opening is binding if the following holds:

$$\Pr\left(\begin{array}{c} \mathsf{PK} \leftarrow \mathsf{Setup}(1^\kappa, t), (\mathcal{C}, \langle B, w_B, r(x)\rangle, \langle i \in B, w_i, \phi(i)\rangle) \leftarrow \mathcal{A}(\mathsf{PK}) : \\ \mathsf{VerifyEvalBatch}(\mathsf{PK}, \mathcal{C}, B, r(x), w_B) = 1 \wedge \\ \mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, \phi(i), w_i) = 1 \wedge \\ \phi(i) \neq r(i) \end{array}\right) = \epsilon(\kappa).$$

Theorem 3.7. *The construction of $\mathsf{CreateWitnessBatch}$, $\mathsf{VerifyEvalBatch}$ is binding provided the t-BSDH assumption holds in $\mathbb{G}$.*

Proof. Suppose an adversary outputs a commitment $\mathcal{C}$ and two openings $\langle B, w_B, r(x)\rangle$, and $\langle i \in B, w_i, \phi(i)\rangle$. We show this adversary can be used to solve the instance of the t-BSDH problem given by the system parameters with non-negligible probability. Let $p(x) = \prod_{i\in B}(x-i)$, and define $p'(x) = p(x)/(x-i)$. Write $w_B = g^b$ and $w_i = g^y$ for some $b, y \in \mathbb{Z}_p$.

Since the verification equations of $\mathsf{VerifyEval}$ and $\mathsf{VerifyEvalBatch}$ hold,

$$e(\mathcal{C}, \hat{g}) = e(w_B, \hat{g}^{p(\alpha)})e(g^{r(\alpha)}, \hat{g}) = e(w_i, \hat{g}^{\alpha-i})e(g, \hat{g}^{\phi(i)})$$

and so from the exponents:

$$\begin{aligned} p(\alpha)b + r(\alpha) &= (\alpha - i)y + \phi(i) \\ p(\alpha)b - (\alpha - i)y &= \phi(i) - r(\alpha) \ . \end{aligned}$$

Now, write $r(\alpha) = r'(\alpha)(\alpha - i) + r(i)$ for $r'(x) = \frac{r(x)-r(i)}{x-i}$, and substitute for $r(\alpha)$, which gives

$$\begin{aligned} p(\alpha)b - (\alpha - i)y &= \phi(i) - r'(\alpha)(\alpha - i) - r(i) \\ (\alpha - i)[p'(\alpha)b - y + r'(\alpha)] &= \phi(i) - r(i) \\ \frac{1}{\alpha - i} &= \frac{p'(\alpha)b - y + r'(\alpha)}{\phi(i) - r(i)} \ . \end{aligned}$$

Note that since $\phi(i) \neq r(i)$ their difference is nonzero, and we can compute $r'(x)$ since we know $r(x)$. Therefore the solution to the t-BSDH problem is

$$e(g, \hat{g})^{\frac{1}{(\alpha - i)}} = e(w_B, \hat{g}^{p'(\alpha)})e(\frac{g^{r'(\alpha)}}{w_i}, \hat{g})^{1/(\phi(i)-r(i))} \ .$$

$\square$

Practicality and Efficiency Improvements. Here, we discuss the practicality of Setup $(1^\kappa, t) = \langle \mathsf{SK}, \mathsf{PK} \rangle$. It is trivial in presence of a trusted authority. In absence of a single trusted party, Setup can be distributed. Here, $\mathsf{SK} = \alpha$ is computed in a distributed form (i.e., shared by multiple parties forming a distributed authority) using the concept of distributed key generation [Ped91b] over $\mathbb{Z}_p$. PK is computed using a distributed multiplication protocol over $\mathbb{Z}_p$ [BOGW88, GRR98]. As we do not require SK during our protocols and as anybody can verify the correctness of PK using pairings, it is possible to consider PK as a global reusable set, shared in many systems.

Further, the exponentiations required when committing and creating witnesses can be trivially parallelized. Also, since $\mathcal{C} = g^{\phi(\alpha)}$ is computed as a product of powers (sometimes called a *multi-exponentiation*), we suggest using fast exponentiation techniques [Pip76] instead of a naïve implementation. It is also possible to precompute $e(\mathcal{C}, \hat{g})$ and $e(g, \hat{g})$ for use during verification.

3.4 Applying **PolyCommit** to VSS

In Section 2.2, we introduced the concept of VSS. Feldman [Fel87] developed the first efficient VSS protocol, which forms the basis of all synchronous VSS schemes defined in the

Sh Phase

1. To share a secret $s \in \mathbb{Z}_p^*$, the dealer P_d chooses a random degree t polynomial $\phi(x) = \sum_{j=0}^{t} \phi_j x^j \in \mathbb{Z}_p[x]$ such that $\phi(0) = \phi_0 = s$. It then broadcasts $\mathcal{C} = \mathsf{Commit}(\mathsf{PK}, \phi(x))$.
2. For $\ell \in [1, n]$, P_d computes a share $s_\ell = \phi(\ell)$, a witness $w_\ell = \mathsf{CreateWitness}(\mathsf{PK}, \phi(x), \ell)$ and sends $\langle \ell, \phi(\ell), w_\ell \rangle$ to node P_ℓ over a secure and authenticated channel.
3. After receiving $\langle i, \phi(i), w_i \rangle$ from P_d, node P_i runs $\mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}, i, \phi(i), w_i)$. If the verification fails, P_i broadcasts an accusation message against P_d.
4. If more than t nodes accuse P_d, then it is clearly faulty and is disqualified. If not, for each accusing party P_ℓ, P_d broadcasts the corresponding share and witness $\langle \ell, \phi(\ell), w_\ell \rangle$ such that VerifyEval holds.
5. If any of the revealed shares fails VerifyEval, P_d is disqualified and the protocol stops. If there is no disqualification, each node P_ℓ accepts the received share s_ℓ.

Rec Phase

During reconstruction, any $t + 1$ or more nodes P_i braodcast their accepted shares and witnesses $\langle i, s_i, w_i \rangle$. All $t + 1$ (or more) nodes verify each of the broadcast shares $\langle i, \phi(i), w_i \rangle$ using VerifyEval and then interpolate the pairs $\langle i, \phi(i) \rangle$ to determine the secret $s = \phi(0)$.

Figure 3.1: eVSS: An efficient Feldman VSS using PolyCommit

literature. In the literature, the discrete logarithm commitment or Pedersen commitment is used in Feldman VSS to achieve the binding (correctness) and the hiding (secrecy) properties. Both of these commitment schemes trivially satisfy the strong correctness (VSS-SC) property of VSS (defined in Section 2.2) by the fact that the size of a commitment to a polynomial $\phi(x) \in \mathbb{Z}_p[x]$ is equal to $\deg(\phi) + 1$, which is $O(n)$ (since for optimal resiliency of $\deg(\phi) = t = \lfloor \frac{n-1}{2} \rfloor$). However, the commitment to a polynomial has to be reliably broadcast to all nodes, which results in a linear-size broadcast for Feldman VSS and their variants and a linear complexity gap between the message and the bit complexities. Our goal is to close this gap using PolyCommit. Next, we apply PolyCommit to existing polynomial-based VSS schemes. By doing so, we reduce the reliable broadcast message size of VSS by a linear factor, making it equal to the message complexity. Although PolyCommit can be used in any univariate polynomial-based VSS scheme, we choose the Feldman VSS for ease of exposition.

Our *efficient Feldman VSS* (eVSS) scheme requires $\mathsf{Setup}(1^\kappa, t)$ of PolyCommit be run once, which outputs $\mathsf{PK} = \langle \mathcal{G}, g, g^\alpha, \ldots, g^{\alpha^t}, \hat{g}, \hat{g}^\alpha \rangle$. Further, as we are working in the synchronous communication model, a *resiliency bound* of $n \geq 2t + 1$ is required for VSS to provide correctness against a t-limited adversary as the $n - t$ honest nodes available during the Sh and Rec phases should at least be equal to $t + 1$ (the required threshold).

In Figure 3.1, we present eVSS that uses the PolyCommit scheme in the Feldman VSS. In the Sh and the Rec phases of the eVSS scheme, the VSS methodology remains exactly the same as that of Feldman VSS except here $t+1$ commitments of the form g^{ϕ_j} for $\phi(x) = \sum_{j=0}^{t} \phi_j x^j$ are replaced by a single polynomial commitment $\mathcal{C} = g^{\phi(\alpha)}$. In addition, along with a share s_i, P_d now sends a witness w_i to node P_i. Overall, the eVSS sharing protocol needs $O(1)$ broadcast instead of $O(n)$ required by the Feldman VSS. Further, in case of multiple accusations, dealer P_d can use the batch opening feature described in Section 3.3.4 to provide a single witness for the complete batch.

Theorem 3.8. *The* eVSS *protocol implements a synchronous VSS scheme with VSS-wS and VSS-SC properties for* $n \geq 2t+1$ *provided the* DLog, t-SDH *and* t-polyDH *assumptions hold in* $\mathcal{G}$.

We need to prove the secrecy, correctness and strong correctness properties of a synchronous VSS scheme. Secrecy and correctness result directly from Theorem 3.5, while Theorem 3.6 provides the strong correctness property. The secrecy provided by eVSS is computational against a t-bounded adversary, and unconditional against a $t-1$ bounded adversary. Share correctness is computational.

3.5 Applying PolyCommit to DKG

In Section 2.3, we introduced the concept of DKG. Using the homomorphic nature of $\mathsf{PolyCommit}$, the eVSS scheme can easily be converted to a DKG protocol. Here, we apply the $\mathsf{PolyCommit}$ commitment scheme to Joint Feldman DKG (JF-DKG) [GJKR07, Section 3] to define a DKG construction that achieves weak correctness and secrecy guarantees (DKG-wC and DKG-wS) defined in Section 2.3.

Our construction ($\mathsf{eJF\text{-}DKG}$) does not guarantee the uniform randomness of the shared secret; it rather creates a shared secret with a hard DLog instance. It is nearly equivalent to running an instance of eVSS protocol (Figure 3.1) by each node in the group.

Here, in order to generate the public key $Y = g^s$, along with the usual commitment $\mathcal{C}$ of an eVSS instance, each node P_i also broadcasts g^{z_i}, w_{i0} and $\mathrm{NIZKPK}_{\mathsf{DLog}}(z_i, g^{z_i})$. For polynomial $\phi_i(x) \in \mathbb{Z}_p[x]$ contributed by node P_i, $z_i = \phi_i(0)$ and $w_{i0} = g^{\psi_{i0}(\alpha)}$ is the corresponding witness with $\psi_{i0}(x) = \frac{\phi_i(x)-\phi_i(0)}{x-0}$. $\mathrm{NIZKPK}_{\mathsf{DLog}}(z_i, g^{z_i})$ is a NIZKPK of the shared value z_i. Upon receiving broadcast from P_ℓ, each node verifies $\langle g^{z_\ell}, w_{\ell 0}, \mathrm{NIZKPK}_{\mathsf{DLog}}(z_\ell, g^{z_\ell})\rangle$. At the end of Sh phase, each node P_i computes the public key $Y = \prod_{P_\ell \in \mathcal{Q}} g^{z_\ell}$, where $\mathcal{Q}$ is the set nodes whose eVSS instances complete without disqualification. Figure 3.2 describes the protocol in detail. The setup remains same as that of the eVSS protocol. Note that in $\mathsf{eJF\text{-}DKG}$ the random oracle assumption is required solely for the NIZKPK. It may be avoided using NIZKPK in the common reference string model [BFM88] or using an interactive ZKPK.

Sh Phase

Generating shared secret s:

1. Each node P_i chooses a random degree t polynomial $\phi_i(x) = \sum_{j=0}^{t} \phi_{ij} x^j \in \mathbb{Z}_p[x]$. Let $\phi_i(0) = \phi_{i0} = z_i$. P_i computes $Y_i = g^{z_i}$, and its witness $w_{i0} = \mathsf{CreateWitness}(\mathsf{PK}, \phi_i(x), 0)$ and a NIZKPK $\pi_{\mathsf{DLog}_i} = \mathrm{NIZKPK}_{\mathsf{DLog}}(z_i, Y_i)$ of z_i. It then broadcasts $\langle \mathcal{C}_i = \mathsf{Commit}(\mathsf{PK}, \phi_i(x)), Y_i, w_{i0}, \pi_{\mathsf{DLog}_i} \rangle$.
2. For $\ell \in [1, n]$, P_i computes a subshare $s_{i\ell} = \phi_i(\ell)$ a witness $w_{i\ell} = \mathsf{CreateWitness}(\mathsf{PK}, \phi_i(x), \ell))$ and sends $\langle \ell, \phi_i(\ell), w_{i\ell} \rangle$ to node P_ℓ over a secure and authenticated channel.
3. After receiving broadcast $\langle \mathcal{C}_\ell, Y_\ell, w_{\ell 0}, \pi_{\mathsf{DLog}_\ell} \rangle$ from P_ℓ, node P_i runs $\mathsf{VerifyEval}'(\mathsf{PK}, \mathcal{C}_\ell, 0, Y_\ell, w_{\ell 0}, \pi_{\mathsf{DLog}_\ell})$ from Section 3.3.4. If the verification fails, then P_ℓ is disqualified.
4. After receiving $\langle i, \phi_\ell(i), w_{\ell i} \rangle$ from P_ℓ, node P_i runs $\mathsf{VerifyEval}(\mathsf{PK}, \mathcal{C}_\ell, i, \phi_\ell(i), w_{\ell i})$. If the verification fails, P_i broadcasts an accusation message against P_ℓ.
5. If more than t nodes accuse P_i, then it is clearly faulty and is disqualified. If not, for each accusing node P_ℓ, P_i broadcasts the corresponding $\langle \ell, s_{i\ell}, w_{i\ell} \rangle$, such that the above verification holds.
6. If any of the shares revealed by P_ℓ fails the verification, P_ℓ is disqualified. If there is no disqualification, each node P_i adds P_ℓ to its qualified nodes set $\mathcal{Q}$.
7. Each node P_i computes its share $s_i = \sum_{P_\ell \in \mathcal{Q}} s_{\ell i}$, witness $w_i = \prod_{P_\ell \in \mathcal{Q}} w_{\ell i}$, commitment $\mathcal{C} = \prod_{P_\ell \in \mathcal{Q}} \mathcal{C}_\ell$ and the public key $Y = \prod_{P_\ell \in \mathcal{Q}} Y_\ell$ of the shared secret $s = \sum_{P_\ell \in \mathcal{Q}} z_\ell$.

Rec Phase

During reconstruction, any $t + 1$ or more nodes P_i braodcast their accepted shares $\langle i, s_i, w_i \rangle$. All $t + 1$ (or more) nodes verify each of the broadcast shares $\langle i, s_i, w_i \rangle$ against $\mathcal{C}$, and then interpolate the pairs $\langle i, s_i \rangle$ to determine the secret s.

Figure 3.2: eJF-DKG: An efficient JF-DKG using PolyCommit

We prove the DKG-wC and DKG-wS properties of DKG (Section 2.3) for the above eJF-DKG protocol.

Theorem 3.9. *The eJF-DKG protocol implements a synchronous DKG scheme with DKG-wC and DKG-wS properties for $n \geq 2t + 1$ provided the DLog, t-SDH and t-polyDH assumptions hold in $\mathcal{G}$.*

Proof. We prove that DKG-wC and DKG-wS properties are hold.

Weak Correctness (DKG-wC). Firstly, we need to prove that there is an efficient algorithm that on input shares from $2t + 1$ nodes and the public information produced by the DKG protocol, output the same unique value s, even if up to t shares are submitted by malicious nodes. This can easily be proven using the strong correctness (VSS-SC) property of eVSS in Theorem 3.8. For a qualified eVSS instance by node P_ℓ, this property assures that any $2t+1$ nodes (or $t+1$ honest nodes) will reconstruct the same z_ℓ shared by P_ℓ. As the secret $s = \sum_{P_\ell \in \mathcal{Q}} z_\ell$, the final share for every node P_i is $s_i = \sum_{P_\ell \in \mathcal{Q}} s_{\ell i}$ and the reconstruction algorithm (Lagrange-interpolation) is homomorphic in nature, shares from any $2t+1$ nodes and the public information result into the same unique value s.

Further, we also need to prove that at the end of Sh phase, all honest nodes have the same value of public key $Y = g^s$, where s the unique secret guaranteed above. Importantly, the set $\mathcal{Q}$ computed by every honest node is the same as it is decided solely based on the broadcast messages. Now to prove that $Y = \prod_{P_\ell \in \mathcal{Q}} g^{z_\ell} = g^{\sum_{P_\ell \in \mathcal{Q}} z_\ell} = g^s$, the only thing that needs to be proven is that g^{z_ℓ} is correct for every $P_\ell \in \mathcal{Q}$. Given $\langle g^{\phi_\ell(0)}, w_{\ell 0}, \mathrm{NIZKPK}_{\mathsf{DLog}}(\phi_\ell(0), g^{\phi_\ell(0)}) \rangle$, we prove this in Section 3.3.4.

Weak Secrecy (DKG-wS). Here, we need to prove that the adversary with t shares and the public key $Y = g^s$ cannot compute the secret s. This follows directly from the secrecy property (VSS-wS) of eVSS in Theorem 3.8, except with slight modifications in the form of creating a witness for the individual contribution z_ℓ by node P_ℓ towards s and a random oracle implementation to simulate NIZKPK of discrete logarithms. $\square$

Our $\mathsf{eJF\text{-}DKG}$ protocol does not achieve the strong correctness and secrecy properties (DKG-C and DKG-S), which is a direct effect of the computational nature of secrecy and correctness of the underlying commitment scheme $\mathsf{PolyCommit}$. In the next chapter, we present a DKG protocol that achieves the strong properties using Pedersen commitments.

Chapter 4

Distributed Key Generation for Use over the Internet

4.1 Preliminaries

As we discussed in the introduction, DKG protocols are the fundamental building blocks of both symmetric and asymmetric threshold cryptography. Numerous applications based on DKG have been proposed; *e.g.*, see [CGS97, NPR99, BF01, GJKR01, Nie02, KKA03, GJKR07, KZG07]. However, the existing DKG protocols assume a synchronous communication model or a (reliable) broadcast channel, which are not guaranteed over the existing Internet [AWL05]. As a result, the systems issues to be considered while realizing DKGs over the Internet have largely been ignored and there is no implementation available. This need for a practical DKG forms the motivation of our work on distributed key generation in the asynchronous communication setting.

In this chapter, we present the first practical DKG protocol [KG09] that we have developed for use over the Internet.

- Section 4.2 defines a realistic system model over the Internet. We combine the standard Byzantine adversary with crash recovery and network failures in an asynchronous setting. We also analyze the asynchronous versus partially synchronous dichotomy for the Internet and justify the choice of treating crashes and network failures separately.
- In Section 4.3, we present a VSS scheme (HybridVSS) that works in our system model. Observing the necessity of a protocol for agreement on a set for asynchronous DKG,

we define and prove a practical DKG protocol in the next section. We use a leader-based agreement scheme in our DKG, as we observe a few pragmatic issues with the usually suggested randomized agreement schemes.

- Finally, in Section 4.5, we discuss the uniform randomness of the shared secret and modify our DKG protocol to achieve uniform randomness in the random oracle model.

We leave discussions about the proactive security and group modification protocols and our implementation and tests over the PlanetLab platform to Chapter 5 and Chapter 8 respectively.

Asynchronous VSS. As we discussed in Section 3.4, Feldman [Fel87] developed the first efficient VSS protocol and Pedersen [Ped91a] presented a modification to it. Nevertheless, these protocols do not guarantee a correct completion in the asynchronous communication model and we need a VSS scheme that works in the asynchronous setting. Before shifting to the asynchronous VSS protocols, it is important to see why the synchronous VSS protocols fail once the synchrony assumption is removed. We choose our eVSS protocol defined in Section 3.4 to demonstrate that. eVSS uses the Feldman VSS methodology [Fel87], which forms the basis of all synchronous VSS schemes defined in the literature; the attack described below can be applied to all these schemes.

In eVSS, as described in Figure 3.1, if the synchrony assumption is violated, then one or more honest nodes might not complete the Sh phase. For example, an accusation broadcast by an honest node might not reach other honest nodes before they complete the Sh phase. This makes the protocol run unreliable, as one or more honest nodes do not complete the Sh phase, while other honest nodes complete the Sh phase assuming that all nodes have successfully completed or would eventually complete the Sh phase. In some scenarios, this may hamper *liveness* or the completion guarantee of the Rec phase as not all $t + 1$ honest nodes have their shares. Further, by breaking the correctness of the broadcast channel, it is also possible for a malicious adversary to break the correctness of the VSS scheme. Note that the secrecy property, however, always remains intact.

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 $\Omega(n^5)$ bit complexities, 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. Of these, the APSS protocol is impractical for any reasonable system size, as it uses a combinatorial secret sharing scheme by Ito, Saito and Nishizeki [ISN87], which leads to an exponential $\binom{n}{t}$ factor in its message complexity. MPSS, on the other hand, is developed for a more mobile setting where set of the system nodes has to change completely between two consecutive phases to maintain the secrecy and correctness properties.

AVSS is the most general and practical scheme in the asynchronous communcation model against Byzantine adversaries, but it does not handle crash recoveries. It assimilates a bivariate polynomial into Bracha's reliable broadcast [Bra84] and can provide complete flexibility with the sets used without hampering the security. In asynchronous VSS, any two participants need to verify the dealer's commitment with each other to achieve correctness; thus, a protocol with $o(n^2)$ message complexity does not seem to be possible and AVSS, with its optimal message complexity, is optimal in that sense. As a result, AVSS forms the basis for our VSS and DKG protocols.

4.2 Hybrid 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. This generic system model will also be applicable to many other distributed protocols over the Internet.

4.2.1 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 *synchronous network* (bounded message delivery delays and processor speeds) and develop more efficient protocols using well-known message delivery time bounds and system run-time assumptions.

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 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 when honest nodes communicate 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 Assumption (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 with adversary nodes, 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.

Note that this assumption is also strictly weaker than the partially synchronous communication assumptions defined by Dwork, Lynch and Stockmeyer [DLS88]). In their first partial synchrony assumption message delivery delays and processor speeds are bounded, but the bounds are not known a priori, while in their second partial synchrony assumption, the bounds are known, but are only guaranteed to hold starting some unknown time. In the asymptotic notation, both of these delay notations are $\Theta(1)$. However, the weak synchrony assumption is $o(t)$, and therefore, weaker than the partial synchrony assumptions.

4.2.2 Byzantine Adversary, Crash-Recoveries and Link Failures

Most of the distributed computing protocols in the literature assume a t-limited *Byzantine adversary* and a compromised node remains Byzantine and unused after recovery. We also aim at proactive security for our DKG (Chapter 5), where the t-limited mobile Byzantine adversary can change its choice of t nodes as time progresses. There, 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 bit 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. A node that is crashed means that *some* of its links are down, not necessarily that they *all* are. 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. In addition to these nodes that crash completely (losing any keys they may have) and subsequently recover from that state, other non-Byzantine nodes may remain up, but have *some* of their links down; those links remain down throughout the protocol (otherwise, they are simply normal asynchronous links), and the nodes may or may not realize that the links are down. These nodes will not need to recover their keys, since they have not lost them. As long as the size of the list of crashed nodes, including all completely crashed nodes, and one of the endpoints of every broken link, does not exceed f at any time, our protocols will succeed. We also assume that the adversary (eventually) delivers all the messages between two uncrashed nodes.

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.

4.2.3 Complexity and Cryptographic Assumptions

Cryptographic Assumptions. Our adversary is computationally bounded with a security parameter κ and it has to break the DLog assumption (Definition 2.5) of size κ-bits to break the security of the protocols. Further, as discussed in Section 2.4, our t-limited adversary is also *static*.

We use a PKI infrastructure in the form of a PKI hierarchy with an external certifying authority (CA) or web-of-trust among nodes to achieve authenticated and confidential communication with TLS links, and message authentication with any digital signature scheme secure against adaptive chosen-message attacks [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.

Complexity Assumptions. In our hybrid model, nodes may crash and recover repeatedly with a maximum of f crashed nodes at any instant. We still need to bound the crashes as an unbounded number of crashes can cause the protocol execution time to be unbounded. We restrict the adversary by a function $d(\cdot)$ that represents the maximum number of crashes that it is allowed to perform during its lifetime and parametrize $d(\cdot)$ with the security parameter κ. Note that, based on the system requirements, it is possible to make $d(\cdot)$ constant or parametrize it with n, t or f.

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 and κ. 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 ϵ_A, such that for all $\kappa \geq 0$, $Pr[X_A(\kappa) > T(\kappa)] \leq \epsilon_A(\kappa)$. 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 bit complexity of a protocol is d-uniformly bounded for some polynomial d.

4.3 Verifiable Secret Sharing for the Hybrid Model

VSS is the fundamental building block for DKG. 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.

4.3.1 Construction: HybridVSS

As usual, our VSS protocol is composed of a sharing protocol (Sh) and a reconstruction protocol (Rec). In protocol Sh, a dealer P_d upon receiving an input message $(P_d, \tau, \mathsf{in}, \mathsf{share}, s)$, shares the secret s, where a counter τ and the dealer identity P_d forms a unique session identifier. Node P_i finishes the Sh protocol by outputting a $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}, s_i)$ message, where $\mathcal{C}$ is the commitment and s_i is its secret share. Any time after that, upon receiving an input message $(P_d, \tau, \mathsf{in}, \mathsf{reconstruct})$, P_i starts the Rec protocol. The Rec protocol terminates for a node P_i by outputting a message $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$, where z_i is P_i's reconstructed value of the secret s. Note that, for the simplicity of discussion, we use DLog commitments instead of the Pedersen commitments used in the original AVSS protocol and achieve VSS-wS secrecy (as defined in Section 2.2). It is easily possible to use Pedersen commitments instead and achieve VSS-S secrecy.

Definition 4.1. *In session (P_d, τ), 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, τ), then all honest finally up nodes complete protocol Sh.*

Agreement. *If some honest node completes protocol Sh of session (P_d, τ), then all honest finally up nodes will eventually complete protocol Sh in session (P_d, τ). If all honest finally up nodes subsequently start protocol Rec for session (P_d, τ), then all honest finally up nodes will finish protocol Rec in session (P_d, τ).*

Correctness. *Once $t + 1$ honest nodes complete protocol Sh of session (P_d, τ), then there exists a fixed value z such that*

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

This is equivalent to the strong correctness (VSS-SC) property defined in Section 2.2.

Secrecy. *If an honest dealer has shared secret s in session (P_d, τ) and no honest node has started the Rec protocol, then, except with negligible probability, the adversary cannot compute the shared secret s. This is equivalent to the VSS-wS secrecy property defined in Section 2.2.*

```
Sh protocol for node $P_i$ and session $(P_d, \tau)$
upon initialization:
  for all $\mathcal{C}$ do
    $A_\mathcal{C} \leftarrow \emptyset$; $e_\mathcal{C} \leftarrow 0$; $r_\mathcal{C} \leftarrow 0$
    $cnt \leftarrow 0$; $cnt_\ell \leftarrow 0$ for all $\ell \in [1, n]$

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

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

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

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

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

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

Figure 4.1: HybridVSS protocol (Sharing step)

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

```
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 $(g^\sigma = \prod_{j=0}^{t} (\mathcal{C}_{j0})^{m^j})$ then
    $S \leftarrow S \cup \{(m, \sigma)\}$; $c \leftarrow c + 1$
    if $c = t + 1$ then
      interpolate constant term $(z_i)$ from $S$; output $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$
```

Figure 4.2: HybridVSS protocol (Reconstruction step)

Figure 4.1 describes the Sh protocol and Figure 4.2 describes the Rec protocol. We use pseudo-code notation and include a special condition **upon** to define actions based on messages received from other nodes or system events. $\mathcal{C}$ is a matrix of commitment entries and $e_{\mathcal{C}}$ and $r_{\mathcal{C}}$ 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_ℓ. Counters *cnt* and cnt_ℓ keep track of the numbers of overall help requests and help requests sent by each node P_ℓ respectively. In presence of crash-recoveries and malicious nodes, a node may receive a message identified by P_d, τ and $\mathcal{C}$ multiple times. According to the protocol definition, it processes the message only the first time it receives it, and ignores subsequent receipts. We use the following predicates in our protocol.

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

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

Note that the AVSS and our HybridVSS schemes use bivariate polynomials, as they guarantee that the interpolated polynomials $\overline{a}$ are of degree t or less. If the univariate polynomials with the constant term equal to the secret s are instead used by dealers in the send messages and the univariate polynomials with the constant term equal to their shares s_i are instead used by nodes in the echo and ready messages, then degrees of the interpolated polynomials $\overline{a}$ will be greater than t with overwhelming probability.

4.3.2 Analysis

The main theorem for $\mathsf{HybridVSS}$ is as follows.

Theorem 4.2. *With the DLog assumption, protocol $\mathsf{HybridVSS}$ implements asynchronous verifiable secret sharing in the hybrid model for $n \geq 3t + 2f + 1$.*

Proof. We need to show liveness, agreement, correctness, secrecy, 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 two lemmas.

Lemma 4.3 (Lemma 1 [BC03]). *Let P_i be a finally up party during session (P_d, τ). Then every distinct message sent to P_i by another finally up party P_j during session (P_d, τ) will be received by P_i in a non-crashed state, provided all associated messages are delivered.*

Lemma 4.4 (Lemma 2 [CKAS02]). *Suppose an honest node P_i sends a ready message containing commitment $\mathcal{C}_i$ and a distinct honest node P_j sends a ready message containing commitment $\mathcal{C}_j$. Then $\mathcal{C}_i = \mathcal{C}_j$.*

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

We assume that the dealer P_d is honest and finally up. According to Lemma 4.3, send messages of the form $(P_d, \tau, \mathsf{send}, \mathcal{C}, a_i)$ 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 $(P_d, \tau, \mathsf{echo}, \mathcal{C}, a_i(j))$ to each system node P_j. Using Lemma 4.3, 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 $\mathsf{verify\text{-}point}$. 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 $(P_d, \tau, \mathsf{ready}, \mathcal{C}, \overline{a}_j(m))$ 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 4.3, each finally up honest node will receive at least $n - t - f$ 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, τ), then all honest finally up nodes will eventually complete protocol Sh during session (P_d, τ). An honest node completes the sharing when it receives $n - t - f$ valid ready messages. At least $t + f + 1$ of those have been sent by honest nodes. Using the definitions of $\mathsf{verify\text{-}poly}$ and $\mathsf{verify\text{-}point}$, the honest node sends only valid ready messages. Further, when sending, an honest node sends ready messages to all system nodes. Thus, using Lemma 4.4, every honest finally up node receives at least $t + f + 1$ valid ready messages with the same commitment $\mathcal{C}$ and sends a ready message containing $\mathcal{C}$. Consequently, every honest finally up node receives $n-t-f$ valid ready messages with commitment $\mathcal{C}$ 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, τ), then all honest finally up nodes will finish protocol Rec during session (P_d, τ) 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 $\mathcal{C}$. Moreover, P_i has received enough valid echo or ready messages with commitment $\mathcal{C}$ and it computes valid ready messages and a valid share s_i with respect to $\mathcal{C}$ (s_i such that $(g^{s_i} = \prod_{j=0}^{t}(\mathcal{C}_{j0})^{m^j})$ 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.

Correctness (VSS-SC). Suppose an honest dealer has shared a degree-t symmetric bivariate polynomial $\phi(x,y)$ with constant term equal to the shared secret s. As the dealer is honest, an echo message that an honest node P_i receives from another honest node P_j contains $\mathcal{C}, \phi(j,i)$. As the required number of 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 $\mathcal{C}'$ different from commitment $\mathcal{C}$ suggested by the dealer. Subsequently, such an honest node P_i, after verification with verify-point, interpolates a polynomial $\overline{a}(y)$ such that $\overline{a}(y) = \phi(i,y)$. Assume an honest node receives $t+1$ ready messages before obtaining $\lceil \frac{n+t+1}{2} \rceil$ commitment $\mathcal{C}$ echo messages. Using Lemma 4.4 all these ready messages have the same commitment and with at least of one of them from an honest node, it is equal to $\mathcal{C}$. The honest node will interpolate the same $\overline{a}(y)$ as in the case of the echo messages. Using the agreement property, if a node completes the protocol 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 $s_{j,d} = \overline{a}(0) = \phi(j,0)$. Let $\lambda_j^{\mathcal{S}}$ be Lagrange interpolation coefficients for the set $\mathcal{S}$ and position 0. We have

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

and if the dealer is honest and has shared secret s during session (P_d, τ), 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 by interpolating 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 $\mathcal{C}$ using Lemma 4.4. As the shares in S_i and S_j are verified against commitment $\mathcal{C}$ and they are valid, it is easy to see that $g^{z_i} = \mathcal{C}_{00} = g^{z_j}$. As g is a generator for a prime order group, $z_i = z_j$.

Secrecy (VSS-wS). To prove the secrecy property, we use the DLog assumption. The adversary's view consists of polynomials $\phi(i, y)$ for these Byzantine nodes i, and the commitment matrix $\mathcal{C}$ 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, $\mathcal{C}$ and t degree-t univariate polynomials consistent with $\mathcal{C}$. We prove that a challenger $\mathcal{B}$ with an oracle access to such an adversary algorithm $\mathcal{A}$ can solve any DLog instance (g, g^α).

Given a DLog instance (g, g^α), the challenger $\mathcal{B}$ generates t degree-t polynomials $r_i(y) \in \mathbb{Z}_p[y]$ and associates them with non-zero indices i. It then computes $g^{r_i(0)}$ for each index i, sets $g^{r_0(0)} = g^\alpha$ and computes $\mathcal{C}_{0,k} = \mathcal{C}_{k,0}$ for $k \in [0, t]$ by interpolation. Proceeding similarly, for each $0 < \ell \leq t$, it uses $\mathcal{C}_{0,\ell}$ and $g^{r_i(\ell)}$ for each index i and computes $\mathcal{C}_{\ell,k} = \mathcal{C}_{k,\ell}$ and completes the symmetric commitment matrix $\mathcal{C}$ which is consistent with g^α as $\mathcal{C}_{0,0}$ and polynomials $r_i(y)$. $\mathcal{B}$ can then present this matrix $\mathcal{C}$ along with polynomials r_i to the adversary algorithm $\mathcal{A}$ and return the output $s = \alpha$ as the DLog value for tuple (g, g^α). As this is not possible, except with negligible probability, we have proven that if an honest dealer has shared secret s during session (P_d, τ) and no honest node has started the Rec protocol, then the adversary cannot compute the secret s except with negligible probability.

Note that, using Pedersen commitments instead of DLog commitments, we can easily achieve and prove the VSS-S property that the adversary has no information about the shared secret s.

Efficiency. Initially, we discuss complexities when there are no crashes. A protocol execution without any crashes has $O(n^2)$ message complexity and $O(\kappa n^4)$ bit complexity where the size of the message is dominated by the commitment matrix $\mathcal{C}$ 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 bit complexity to $O(\kappa n^3)$. In this approach, commitments are generated using the exponentiated form of bivariate polynomial evaluations ($A_j^{(i)} = g^{\phi(i,j)}$). Let $A^{(j)} = \langle A_0^{(j)}, A_1^{(j)}, \ldots, A_n^{(j)} \rangle$. In this case, the bit complexity gets reduced by a linear factor using the $A^{(0)}$ vector and a vector $h = \langle h_1, \ldots, h_n \rangle$, where H is collision-resistant hash function and $h_j = H(A^{(j)})$.

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 help messages, out of which first $d(\kappa)$ will be answered by honest nodes. Therefore, each honest node will in total answer up to $(t + 1)d(\kappa)$ 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 bit complexity of $\mathsf{HybridVSS}$ are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively and we obtain a uniform polynomial bound on the bit complexity. $\square$

PolyCommit to Asynchronous VSS. The above VSS uses commitments of size $\Theta(\kappa n)$ in the case of commitment vectors and $\Theta(\kappa n^2)$ in the case of commitment matrices. This leads to bit complexities of $\Theta(\kappa n^3)$ and $\Theta(\kappa n^4)$ respectively. We are currently working on a bivariate version of our PolyCommit scheme defined in Section 3.3, which with its constant size commitments, may reduce this bit complexity further to $\Theta(\kappa n^2)$.

4.4 Distributed Key Generation for the Hybrid Model

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] solve a similar agreement problem in their proactive refresh protocol using a multi-valued validated Byzantine agreement (MVBA) protocol. Known expected constant-round MVBA protocols [CKPS01] require threshold signature and threshold coin-tossing primitives [CKS00]. The algorithms suggested for both of these primitives in [CKS00] require either a dealer or a DKG. As we aim to avoid the former (dealer) in this work and the latter (DKG) is our aim itself, we cannot use their MVBA protocol.

In more technical terms, randomization in the form of distributed coin tossing or equivalent randomization functionality is necessary for an expected constant-round Byzantine agreement such as MVBA [CKPS01]; it thwarts the attack possible with an adversary knowing the pre-defined node selection order by making completely random selections. However, an efficient algorithm for dealerless distributed coin tossing without a DKG is difficult to achieve. Canetti and Rabin [CR93] define a dealerless distributed coin tossing protocol without DKG; however, their protocol requires n^2 VSSs for each coin toss and is consequently inefficient. Therefore, we refrain from using randomized agreement.

We follow a much simpler approach with the same bit 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 (HybridDKG).

4.4.1 Construction: **HybridDKG**

In HybridDKG, for session τ and leader $\mathcal{L}$, each node P_d selects a secret value s_d and shares it among the group using protocol Sh of HybridVSS for session (P_d, τ). Each node finishes the HybridDKG protocol by outputting a $(\overline{\mathcal{L}}, \tau, \text{DKG-completed}, \mathcal{C}, s_i)$ message, where s_i and $\mathcal{C}$ are its share and the commitment respectively and $\overline{\mathcal{L}}$ is the finally agreed upon leader.

Definition 4.5. *In session τ, protocol HybridDKG 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 HybridDKG in session τ, except with negligible probability.*

Agreement. *If some honest node completes protocol HybridDKG in session τ, then, except with negligible probability, all honest finally up nodes will eventually complete protocol HybridDKG in session τ.*

Correctness. *Once an honest node completes the HybridDKG protocol for session τ, then there exists a fixed value s such that, if an honest node P_i reconstructs z_i in session τ, then $z_i = s$. This is equivalent to the weak correctness (DKG-wC) property defined in Section 2.3.*

Secrecy. *If no honest node has started the Rec protocol, then, except with negligible probability, the adversary cannot compute the shared secret s. This is equivalent to the DKG-wS secrecy property defined in Section 2.3.*

Efficiency. *The bit complexity for any instance of HybridDKG is d-uniformly bounded.*

We first describe the optimistic phase of our HybridDKG protocol. For each session τ, one among n nodes works as a leader. The leader $\mathcal{L}$, once it finishes the VSS proposal by $t + 1$ nodes with $(P_d, \tau, \text{out}, \text{shared}, \mathcal{C}_d, s_{i,d})$, 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 $(P_d, \tau, \text{out}, \text{shared}, \mathcal{C}_d, s_{i,d}, \mathcal{R}_d)$, where a set $\mathcal{R}_d$ includes $n - t - f$ signed ready messages for session (P_d, τ). Once this

```
**Optimistic phase for node $P_i$ in Session ($\tau$) with Leader $\mathcal{L}$**
upon initialization:
  $e_{\mathcal{L},\mathcal{Q}} \leftarrow 0$; $r_{\mathcal{L},\mathcal{Q}} \leftarrow 0$ 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 $\mathcal{L}$
  $cnt \leftarrow 0$; $cnt_\ell \leftarrow 0$ for all $\ell \in [1,n]$; $lc_{\mathcal{L}} \leftarrow 0$ for each $\mathcal{L}$; $lc_{flag} \leftarrow$ false ; $\mathcal{L}_{next} \leftarrow \mathcal{L}+n-1$
  for all $d \in [1,n]$ do
    initialize extended-HybridVSS Sh protocol $(P_d, \tau)$
upon $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}_d, s_{i,d}, \mathcal{R}_d)$ (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$(\mathcal{Q}, \mathcal{R}/\mathcal{M})$ and $(\overline{\mathcal{Q}} = \emptyset$ 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):
  $e_{\mathcal{L},\mathcal{Q}} \leftarrow e_{\mathcal{L},\mathcal{Q}} + 1$
  if $e_{\mathcal{L},\mathcal{Q}} = \lceil \frac{n+t+1}{2} \rceil$ and $r_{\mathcal{L},\mathcal{Q}} < t+1$ 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):
  $r_{\mathcal{L},\mathcal{Q}} \leftarrow r_{\mathcal{L},\mathcal{Q}} + 1$
  if $r_{\mathcal{L},\mathcal{Q}} = t+1$ and $e_{\mathcal{L},\mathcal{Q}} < \lceil \frac{n+t+1}{2} \rceil$ 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 $r_{\mathcal{L},\mathcal{Q}} = n-t-f$ then
    stop timer, if any; wait for shared output-messages for each $P_d \in \mathcal{Q}$
    $s_i \leftarrow \sum_{P_d \in \mathcal{Q}} s_{i,d}$; $\forall_{p,q} : \mathcal{C}_{p,q} \leftarrow \prod_{P_d \in \mathcal{Q}} (\mathcal{C}_d)_{p,q}$; output $(\mathcal{L}, \tau, \mathsf{DKG\text{-}completed}, \mathcal{C}, s_i)$
upon timeout:
  if $lc_{flag} =$ false then
    if $\overline{\mathcal{Q}} = \emptyset$ then
      $lc_{flag} \leftarrow$ true; send msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}+1, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})_{\mathsf{sign}}$ to each $P_j$
    else
      $lc_{flag} \leftarrow$ true; send msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}+1, \overline{\mathcal{Q}}, \overline{\mathcal{M}})_{\mathsf{sign}}$ to each $P_j$
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 $cnt_\ell \le d(\kappa)$ and $cnt \le (t+1)d(\kappa)$ then
    $cnt_\ell \leftarrow cnt_\ell + 1$; $cnt \leftarrow cnt + 1$; send all messages of $B_{\ell(\mathcal{L},\tau)}$
```

Figure 4.3: HybridDKG protocol (Optimistic phase)

```
**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 $\mathsf{verify\text{-}signature}(\mathcal{Q}, \mathcal{R}/\mathcal{M})$ then
    $lc_{\overline{\mathcal{L}}} \leftarrow lc_{\overline{\mathcal{L}}} + 1$; $\mathcal{L}_{next} \leftarrow \min(\mathcal{L}_{next}, \overline{\mathcal{L}})$
    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 $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{next}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to each $P_j$
      else
        send the msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{next}, \overline{\mathcal{Q}}, \overline{\mathcal{M}})$ 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 $\overline{\mathcal{L}}$
      $\mathcal{L} \leftarrow \overline{\mathcal{L}}$; $lc_{\mathcal{L}} \leftarrow 0$; $\mathcal{L}_{next} \leftarrow \mathcal{L} - 1$; $lc_{flag} =$ 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 4.4: HybridDKG protocol (Pessimistic phase)

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 or slow 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 and provide liveness without damaging system safety. Without loss of generality, we assume that the permutation is a linear sorted order of node indices. Every unsatisfied node sends a signed leader-change (lead-ch) request to all the nodes for the next leader $\mathcal{L}+1$ 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 Section 4.2.1. When a node collects $t+1$ lead-ch messages for leaders $\mathcal{L}+\delta$ for small positive integers δ, 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 enters into the optimistic phase by sending a send message for set $\overline{\mathcal{Q}}$ if it is non-empty or else for set $\widehat{\mathcal{Q}}$. Set $\widehat{\mathcal{Q}}$ contains the indices of nodes whose VSS instances have completed at one more nodes, while set $\overline{\mathcal{Q}}$ is a set of nodes broadcast by the current or a previous leader such that an honest node might have completed that broadcast. In case of leader change, set $\overline{\mathcal{Q}}$ avoids two honest nodes finishing with two different VSSs sets with two different leaders. Once a node receives $\lceil \frac{n+t+1}{2} \rceil$ echo messages or $t+1$ ready messages, $\widehat{\mathcal{Q}}$ is assigned to $\overline{\mathcal{Q}}$ as some or all honest nodes might complete the broadcast even if other times out. $\overline{\mathcal{Q}}$ ensures that the new leader broadcasts the same set $\widehat{\mathcal{Q}}$ and all honest nodes delivers the same set in the agreement. Set $\overline{\mathcal{M}}$ attached to $\overline{\mathcal{Q}}$ contains $\lceil \frac{n+t+1}{2} \rceil$ signed echo messages or $t+1$ signed ready messages for $\overline{\mathcal{Q}}$. Along with condition ($|\widehat{\mathcal{Q}}| = t+1$ and $\overline{\mathcal{Q}} = \emptyset$), set $\overline{\mathcal{M}}$ avoids wrong broadcast 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_ℓ. Counters cnt and cnt_ℓ keep track of the numbers of overall help requests and help requests sent by each node P_ℓ respectively. Figures 4.3 and 4.4 present the optimistic and the pessimistic phases of the HybridDKG protocol. Protocol HybridDKG-Rec remains exactly the same as HybridVSS Rec protocol in Figure 4.2. We denote these Sh and Rec as HybridDKG-Sh$_{\mathsf{DLog}}$ and HybridDKG-Rec$_{\mathsf{DLog}}$. For node P_i, they are declared as follows.

$$\left(\mathcal{C}^{(s)}_{\langle g \rangle}, s_i\right) = \mathsf{HybridDKG\text{-}Sh}_{\mathsf{DLog}}(n, t, f, \tilde{t}, g, \alpha_i) \tag{4.1}$$

$$s = \mathsf{HybridDKG\text{-}Rec}_{\mathsf{DLog}}(t, \mathcal{C}^{(s)}_{\langle g \rangle}, s_i) \tag{4.2}$$

Here, $\tilde{t}$ is the number of VSS instances to be chosen ($t < \tilde{t} \leq 2t+1$), $g \in \mathbb{G}$ is a commitment generator, $\alpha_i \in \mathbb{Z}_p$ is a secret shared by P_i, and $\mathcal{C}^{(s)}_{\langle g \rangle} = [g^s, g^{\phi(1)}, \cdots, g^{\phi(n)}]$ is the discrete logarithm commitment vectors for $\phi \in \mathbb{Z}_p[x]$ of degree t with $s = \phi(0)$ and $s_i = \phi(i)$.

4.4.2 Analysis

The main theorem for our HybridDKG is as follows.

Theorem 4.6. *With the DLog assumption, protocol HybridDKG provides an asynchronous distributed key generation mechanism in the hybrid model for $n \geq 3t + 2f + 1$.*

We need to show the liveness, agreement, correctness, secrecy, and efficiency of our HybridDKG 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 $t+1$ HybridVSS instances. If the leader is honest and uncrashed, it also completes $t+1$ HybridVSS instances, 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 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 lead-ch request is broadcast by that honest node (pessimistic phase). After receiving $n-t-f$ lead-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 HybridDKG will complete. The requirement of $n-t-f$ lead-ch requests for a leader replacement makes sure that nodes do not complete the leader-change too soon. An honest node sends a signed lead-ch message for the smallest leader (among the received set) if it receives $t+1$ lead-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 HybridDKG 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 the 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 HybridDKG then all honest finally up nodes will eventually complete the HybridDKG protocol.

Correctness (DKG-wS). According to the agreement property, once an honest node completes the HybridDKG for session (τ), then all $(n-t-f)$ honest finally up nodes will eventually complete the HybridDKG protocol. According to the correctness 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 the 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 HybridDKG finishes and as Lagrange-interpolation is homomorphic over addition, on reconstruction after the HybridDKG protocol each node will output the same $z_i = s$.

Secrecy (DKG-wS). In HybridDKG 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 the 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 HybridDKG 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 bit complexities of HybridVSS are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively. In the HybridDKG 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 bit complexities of the possible n HybridVSS Sh protocols in HybridDKG are $O(tdn^3)$ and $O(\kappa tdn^4)$ respectively. If the HybridDKG protocol completes without entering into the pessimistic phase, then the system only needs one reliable broadcast of message of size $O(\kappa n)$, message complexity $O(tdn^2)$ and bit complexity $O(\kappa tdn^3)$. As a result, the optimal message and bit complexities for the HybridDKG 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 HybridDKG completes and worst case message and bit complexities of the HybridDKG 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 share refresh protocol for AVSS [CKAS02].

4.5 Achieving Uniform Randomness in HybridDKG

The shared secret in the above HybridDKG may not be *uniformly* random; this is a direct effect of using only the discrete logarithm commitments (see [GJKR07, Section 3] for details). In many cases, we do not need a uniformly random secret key; the security of these schemes relies on the assumption that the adversary cannot compute the secret. However, a uniformly random shared secret may be required in some protocols. In that case, we use Pedersen commitments, but we do not employ the methodology defined by Gennaro *et al.* [GJKR07], which increases the latency in the system. We observe instead that with the random oracle assumption, the communicationally demanding technique by Gennaro *et al.* can be replaced with the much simpler NIZK proof of equality of committed values ($\text{NIZKPK}_{\equiv Com}$) described in Equation 2.2 in Section 2.7. Here, given a discrete logarithm commitment $\mathcal{C}_{\langle g \rangle}(s)$ and a Pedersen commitment $\mathcal{C}_{\langle g,h \rangle}(s, s')$ for $s, s' \in \mathbb{Z}_p$, a prover proves that she knows s and s' such that $\mathcal{C}_{\langle g \rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h \rangle}(s, s') = g^s h^{s'}$. We represent $\mathsf{HybridDKG}$ protocols using Pedersen commitments as $\mathsf{HybridDKG}_{\mathsf{Ped}}$. For node P_i, the corresponding $\mathsf{HybridDKG\text{-}Sh}$ and $\mathsf{HybridDKG\text{-}Rec}$ schemes are defined as follows.

$$\left(\mathcal{C}^{(s,s')}_{\langle g,h \rangle}, [\mathcal{C}^{(s)}_{\langle g \rangle}, \text{NIZKPK}_{\equiv Com}], s_i, s'_i\right) = \mathsf{HybridDKG\text{-}Sh}_{\mathsf{Ped}}(n, t, f, t', g, h, \alpha_i, \alpha'_i) \quad (4.3)$$

$$s = \mathsf{HybridDKG\text{-}Rec}_{\mathsf{Ped}}(t, \mathcal{C}^{(s,s')}_{\langle g,h \rangle}, s_i, s'_i) \quad (4.4)$$

Here, t' is the number of VSS instances to be chosen ($t < t' \leq 2t+1$), $g, h \in \mathbb{G}$ are commitment generators, $\alpha_i, \alpha'_i \in \mathbb{Z}_p$ are respectively a secret and randomness shared by P_i, and $\mathcal{C}^{(s)}_{\langle g \rangle} = [g^s, g^{\phi(1)}, \cdots, g^{\phi(n)}]$ and $\mathcal{C}^{(s,s')}_{\langle g,h \rangle} = [g^s h^{s'}, g^{\phi(1)} h^{\phi'(1)}, \cdots, g^{\phi(n)} h^{\phi'(n)}]$ are respectively the discrete logarithm and Pedersen commitment vectors for $\phi, \phi' \in \mathbb{Z}_p[x]$ of degree t with $\phi(0) = s$ and $\phi'(0) = s'$. The optional $\text{NIZKPK}_{\equiv Com}$ is a vector of zero-knowledge proofs of knowledge that the corresponding entries of $\mathcal{C}^{(s)}_{\langle g \rangle}$ and $\mathcal{C}^{(s,s')}_{\langle g,h \rangle}$ commit to the same values.

In the $\mathsf{HybridDKG}_{\mathsf{Ped}}$ protocol, we use Pedersen commitments in $\mathsf{HybridVSS}$. At the end, each node adds the discrete logarithm commitment of its share and the corresponding $\text{NIZKPK}_{\equiv Com}$ in its $\mathsf{DKG\text{-}completed}$ message and sends this $\mathsf{DKG\text{-}completed}$ message removing the share to each of the other nodes. $\mathsf{HybridDKG}_{\mathsf{Ped}}$ achieves the same liveness and agreement guarantees as those of $\mathsf{HybridDKG}_{\mathsf{DLog}}$, while for correctness and secrecy it respectively achieves DKG-C and DKG-S properties instead of their weaker version in $\mathsf{HybridDKG}_{\mathsf{DLog}}$.

Proof. The liveness and agreement proofs are the same as those of $\mathsf{HybridDKG}_{\mathsf{DLog}}$.

Correctness (DKG-C). For DKG-C, we need to prove the following three properties.

1. There is an efficient algorithm that on input shares from $2t+1$ nodes and the public information produced by the $\mathsf{HybridDKG}$ protocol, output the same unique value s, even if up to t shares are submitted by malicious nodes.

2. At the end of Sh phase of $\mathsf{HybridDKG}_{\mathsf{Ped}}$, all honest nodes have the same value of public key $Y = g^s$, where s the unique secret guaranteed above.

3. s and Y are uniformly distributed in $\mathbb{Z}_n$ and $\mathbb{G}$ respectively.

The first two properties are the same as those in $\mathsf{HybridDKG}_{\mathsf{DLog}}$ and we only need to prove the third property.

Algorithm for Simulator $\mathcal{S}$

Let $\mathcal{B}$ be the set of parties controlled by the adversary, and $\mathcal{G}$ be the set of honest parties (run by the simulator). Without loss of generality, $\mathcal{B} = [P_1, P_{t'}]$ and $\mathcal{G} = [P_{t'+1}, P_n]$, where $t' \leq t$. Let $Y \in \mathbb{G}$ be the input public key and $\mathrm{H}_{\equiv Com} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash table for $\mathrm{NIZKPK}_{\equiv Com}$.

1. Perform all steps on behalf of the uncorrupted parties $P_{t'+1}, \ldots, P_n$ exactly as in the HybridDKG protocol until the DKG-completed message. Once a node is ready to sent the DKG-completed message, the following holds:
   - Set $\overline{\mathcal{Q}}$ is well defined with at least one honest node in it.
   - The adversary's view consists of polynomials $\phi^{(j)}(x, y)$ for $j \in \mathcal{B}$, the share polynomials $a_j^{(i)} y = \phi^{(i)}(j, y)$ for $P_i \in \overline{\mathcal{Q}}$, $P_j \in \mathcal{B}$, and commitments $\mathcal{C}_i$ for $P_i \in \mathcal{Q}$.
   - $\mathcal{S}$ knows all polynomials $\phi^{(i)}(x, y)$ for $P_i \in \overline{\mathcal{Q}}$ as it knows $n - t'$ shares for each of those.
2. Perform the following computations for each $i \in [t + 1, n]$ before starting Step 6:
   (a) Compute s'_j for $P_j \in [1, n]$ and s_j for $P_j \in \mathcal{B}$. Interpolate (in exponent) $(0, Y)$ and (j, g^{s_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g \rangle}(s_i^*) = g^{s_i^*}$.
   (b) Compute the corresponding $\mathrm{NIZKPK}_{\equiv Com}$ by generating random challenge $c_i \in_R \mathbb{Z}_p$ and responses $u_{i,1}, u_{i,2} \in_R \mathbb{Z}_p$, computing the commitments $t_{i,1} = (g^{s_i^*})^{c_i} g^{u_{i,1}}$ and $t_{i,2} = \frac{\mathcal{C}_{\langle g,h \rangle}(s_i, r_i)}{\mathcal{C}_{\langle g \rangle}(s_i^*)}^{c_i} h^{u_{i,2}}$ and include entry $\langle (g, h, \mathcal{C}_{\langle g \rangle}(s_i^*), \mathcal{C}_{\langle g,h \rangle}(s_i, r_i), t_{i,1}, t_{i,2}), c_i \rangle$ in the hash table $\mathrm{H}_{\equiv Com}$ so that $\pi_{\mathsf{DLog}_n} = (c_i, u_{i,1}, u_{i,2})$.
3. In the end, $s = \sum_{P_i \in \overline{\mathcal{Q}}} \alpha_i$ such that $Y = g^s$.

Figure 4.5: Simulator for HybridDKG with the uniform randomness property

Here, $s = \sum_{P_i \in \overline{\mathcal{Q}}} \alpha_i$. As long as there is one value α_i in this sum that is chosen at random and independently from the other values in the sum, the uniform distribution of s is guaranteed. All α_i values are only available in the form a Pedersen commitment until set $\overline{\mathcal{Q}}$ is finalized. From Theorem 4.4 of [Ped91b], in VSS using the Pedersen commitments, the view of the t-limited adversary is independent of the shared secret. Therefore, with at least one VSS from the honest nodes in the $t + 1$ chosen VSSs, s is uniformly distributed and so is $Y = g^s$.

Secrecy (DKG-S). We need to prove that no information about s can be learned by the adversary except for what is implied by $Y = g^s$. More formally, we prove that for every PPT adversary $\mathcal{A}$ that has up to t nodes, there exists a PPT simulator $\mathcal{S}$ that on input $Y \in \mathbb{G}$ produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a run of the HybridDKG protocol that ends with Y as its public key. Our proof is based on the proof of secrecy in [GJKR07, Section 4.3].

In Figure 4.5, we describe the simulator $\mathcal{S}$ for our HybridDKG protocol. An informal description is as follows. $\mathcal{S}$ runs a HybridDKG instance on behalf of all honest nodes. For the most of the protocol (until message DKG-completed is to be sent), it follows the protocol

HybridDKG as instructed. For DKG-completed messages, it changes the public key shares $Y_i = g^{s_i}$ to "hit" the desired public key Y. $\mathcal{S}$ knows all g^{s_j} and $g^{s'_j}$ values for all $P_j \in \mathcal{B}$, as it chooses $\phi^{(j)}(x,y)$ for good nodes and has received enough shares from bad nodes to reconstruct the bivariate polynomials shared by them. For $i \in [t+1, n]$, $\mathcal{S}$ sets $g^{s_i^*}$ as interpolation (in exponent) of $(0, Y)$ and (j, g^{s_j}) for $j \in [1,t]$. It creates the corresponding $\text{NIZKPK}_{\equiv Com}$ using the random oracle hash table.

We show that the view of the adversary $\mathcal{A}$ that interacts with $\mathcal{S}$ on input Y is the same as the view of $\mathcal{A}$ that interacts with the honest nodes in a regular run of the protocol that outputs the given Y as the public key.

In a regular run of protocol HybridDKG, $\mathcal{A}$ sees the following probability distribution of data produced by the honest nodes:

- Values $\phi_i(j,y)$, ${\phi'}_i(j,y)$ for $i \in \mathcal{G}, j \in \mathcal{B}$, uniformly chosen in $\mathbb{Z}_p$
- Values $\mathcal{C}_i$ and g^{s_i} for $P_i \in \mathcal{G}$, that correspond to randomly chosen polynomials.

As we are interested in runs of HybridDKG that end with Y as the public key, we note that the above distribution of values is induced by the choice (of the good parties) of polynomials $\phi_i(x,y)$, ${\phi'}_i(x,y)$ for $P_i \in \overline{\mathcal{Q}}$, uniformly distributed in the family of t-degree polynomials over $\mathbb{Z}_p$ such that $\prod_{P_i \in \overline{\mathcal{Q}}} g^{\phi_i(0,0)} = Y$. Without loss of generality, assume $P_n \in \mathcal{G}$ belongs to $\overline{\mathcal{Q}}$. The above distribution is characterized by the choice of polynomials $\phi_i(x,y)$, $\phi_i^*(x,y)$ for $P_i \in (\mathcal{G} \cap \overline{\mathcal{Q}}) - \{P_n\}$ as random independent t-degree bivariate polynomials over $\mathbb{Z}_p$ and of $\phi_n(x,y)$ as a uniformly chosen polynomial from the family of t-degree bivariate polynomials over $\mathbb{Z}_p$ that satisfy the constraint $\phi_n(0,0) = s - \sum_{P_i \in \overline{\mathcal{Q}} \setminus \{n\}} \phi_i(0,0)$.

We show that the simulator $\mathcal{S}$ outputs a probability distribution which is *identical* to the above distribution. First note that the above distribution depends on the set $\overline{\mathcal{Q}}$ decided as the broadcast by the current leader is complete. Since all actions of the simulator until $\overline{\mathcal{Q}}$ is (eventually) delivered to all nodes are identical to the actions of honest parties interacting with $\mathcal{A}$ in a real run of the protocol, we are assured that the set $\overline{\mathcal{Q}}$ defined in this simulation is identical to its value in the real protocol.

We now describe the output distribution of $\mathcal{S}$ in terms of t-degree bivariate polynomials ϕ_i^* corresponding to the choices of the simulator. It is defined as follows: For $P_i \in (\overline{\mathcal{Q}} - \mathcal{B} - \{P_n\})$, set ϕ_i^* to ϕ_i and ${\phi'}_i^*$ to ${\phi'}_i$. Define ϕ_n^* such that the values $\phi_n^*(0,0) = \log_g\left(\frac{Y}{\prod_{j \in (\overline{\mathcal{Q}} - \mathcal{B} - \{P_n\})} g^{\alpha_j^*}}\right)$ and $\phi_n^*(j,y) = \phi_n(j,y)$ for $j \in [1,t]$. Finally, define ${\phi'}_n^*(x,y)$ such that $\phi_n^*(x,y) + \Lambda {\phi'}_n^*(x,y) = \phi_n(x,y) + \Lambda {\phi'}_n(x,y)$, where $\Lambda = \log_g(h)$. It can be seen by this definition that the univariate polynomial evaluations of these polynomials evaluated at the indices for $P_j \in \mathcal{B}$ coincide with the values $\phi_i(j,y)$ which are seen by the corrupted parties in the protocol. Note that the above DLog values $\phi_n^*(0,0)$ and ${\phi'_n}^*(0,0)$

are unknown to the simulator. Also, the commitments of these polynomials agree with $\mathcal{C}_i$ published by the simulated honest parties in the protocol as well as with the exponentials $g^{s_i^*}$ for $P_i \in \mathcal{G}$ published by the simulator at the end on behalf of the honest parties. Thus, these values pass the verifications in the real protocol.

It remains to be shown that polynomials ϕ_i^* and ϕ'^*_i belong to the right distribution. Indeed, for $\overline{\mathcal{Q}} - \mathcal{G} - \{P_n\}$ this is immediate since they are defined identically to ϕ_i which are chosen according to the uniform distribution. For ϕ_n^* we see that this polynomial evaluates in points $j = [1, t]$ to random values $(\phi_n(j, y))$ while at 0 it evaluates $\log_g(g^{\alpha_n^*})$ as required to hit Y. Finally, ϕ'^*_n is defined as $\phi'^*_n(x, y) = \Lambda^{-1}(\phi_n(x, y) - \phi_n^*(x, y) + \phi'_n(x, y))$ and since $\phi'^*_n(x, y)$ is random and independent then so is $\phi'^*_n(x, y)$. $\square$

Chapter 5

Proactive Security and Group Modification Protocols

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.

Further, 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.

In this chapter, we provide the proactive security and group modification protocols for our HybridDKG protocol presented in Chapter 4. In Section 5.1, we introduce the notion of phase in our hybrid model defined in Section 4.2. In Section 5.2, we present the protocol for share renewal and recovery protocols. Along with proactive security, observing the importance of group modifications for a long-term system sustainability, we devise and prove protocols for group modification agreement, node addition, node removal and threshold and crash-limit modification in Section 5.3. Finally, in Section 5.4, we briefly discuss the utility of our constant-size PolyCommit commitment to the synchronous proactive VSS scheme of Herzberg *et al.* [HJKY95].

5.1 Hybrid System Model for Proactive Security

In this section, we describe the proactive system model. We discuss the concept of common phase, adversary behaviour and forward secrecy for our proactive protocols.

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/secrecy 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 are no shares available at honest nodes until the renewal protocol completes and there is no phase overlap.

Byzantine Adversary. The adversary can corrupt at most t nodes in any local phase $\tau \geq 0$ of any honest party. 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 the crash-recovery and link failure assumption; we justify it in Section 5.2.1. 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. [DvOW92] 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 τ of the sender is delivered to the receiver in the same local phase or it is lost.

5.2 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 design share renewal and recovery protocols.

5.2.1 Share Renewal Protocol

A share renewal protocol enables DKG nodes to renew their shares such that protocol Rec will output the same secret and the adversary does not learn anything about it. From a share renewal protocol, we expect liveness, correctness, secrecy and efficiency similar to the HybridDKG protocol, under the assumption that the adversary delivers all associated messages within phase τ.

Definition 5.1. *Suppose nodes hold shares of a secret s shared using a HybridDKG instance for a phase $\tau - 1$. An asynchronous* share renewal *protocol (Renew) for phase τ 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 τ, then all honest nodes complete the Renew protocol, except with negligible probability.*

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

Secrecy. *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.*

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

- On receiving a clock tick for phase τ, instead of running HybridVSS for a random key, node P_i reshares its share $s_{i,\tau-1}$. It then erases the old share, the bivariate polynomial used during resharing, and the univariate polynomials from the send messages, and broadcasts its clock tick. While retransmitting send messages during a node recovery, only the commitments are sent.
- A node waits for $t+1$ identical clock ticks before proceeding with protocol Sh instances.
- Once a node P_i receives $n-t-f$ 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 $\ell \in [0,t]$.

Analysis. The main theorem for protocol Renew is as follows.

Theorem 5.2. *With the DLog assumption, the Renew protocol implements asynchronous verifiable share renewal in the hybrid model for $n \geq 3t+2f+1$.*

We need to show liveness, correctness, secrecy, and efficiency.

Liveness. Expect for a small modification to maintain forward secrecy, the liveness analysis is exactly same as that for the HybridDKG protocol. We delete the univariate polynomials from the 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 $\lceil (n+t+1)/2 \rceil$ echo messages received during an Sh instance of HybridVSS, each honest node need only receive $t+1$ consistent shares of its univariate polynomial in order that the protocol Sh can continue. With the assumption that at least $t+1$ honest and uncrashed nodes receive the send messages transmitted by an honest and uncrashed node before the adversary can crash them, liveness is guaranteed for each such Sh instance. The remaining liveness discussion remains exactly the same as that of the HybridDKG protocol.

Correctness. According to the correctness property of protocol HybridDKG, for each of the completed sharings, if run individually, each node P_i will reconstruct the same shared secret $z_{i,d}$ where $P_d \in \mathcal{Q}$ 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 HybridDKG, 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 τ 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 τ before detecting a subsequent clock tick, the system maintains a verifiable sharing of s.

Secrecy. Secrecy is proven in exactly the same way as the secrecy of our HybridDKG protocol.

Efficiency. The efficiency discussion remains similar to the efficiency discussion in the HybridDKG 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.

5.2.2 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 system-level techniques beyond the scope of this paper. 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, HybridDKG and Renew protocols suffice to handle share recoveries. To achieve automatic share recovery upon reboot, we add a recover message to nodes' reboot procedure.

5.3 Group Modification Protocols

Here, we present protocols to achieve node addition, node removal, and security-threshold and crash-limit modification.

5.3.1 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 the 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 for different nodes 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. 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.

5.3.2 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.

Definition 5.3. *Suppose nodes hold shares of a secret s shared using a* HybridDKG *instance for a phase τ. A* node addition *protocol (*NodeAdd*) to add a new honest node P_{new} during phase τ 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 τ, then the new honest node P_{new} completes the protocol* NodeAdd *except with negligible probability.*

Correctness. *Once node P_{new} completes protocol* NodeAdd *during phase τ, then there exists a fixed value z such that if P_{new} reconstructs z_i during phase τ, then $z_i = s$.*

Secrecy. *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.*

Construction. We obtain this by making three modifications to our HybridDKG protocol.

- 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 the same commitment vector V_ℓ for $\ell \in [0,t]$, interpolates them for index 0 to obtain its share s_{new}.

Analysis. The main theorem for protocol NodeAdd is as follows.

Theorem 5.4. *With the* DLog *assumption, 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, correctness, secrecy, and efficiency. The liveness and efficiency analysis exactly mirrors that of the HybridDKG 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 correctness and secrecy analysis are quite interesting.

Correctness. According to the correctness property of the reliable broadcast protocol, each of the 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_ℓ for $\ell \in [0,t]$ and protocol Rec at node *new* will reconstruct shared secret s. 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.

Secrecy. 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, secrecy can proven exactly in the same way as secrecy in our HybridDKG 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.

5.3.3 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 $n \geq 3t + 2f + 1$.

5.3.4 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 updated parameters. As a feature of protocol Renew, t and f can be easily changed by just correctly changing the degrees of the resharing polynomials.

5.4 PolyCommit to Proactive Schemes

As we are not able to use our PolyCommit commitments in AVSS [CKAS02], we are not able to use it in the above proactive and group modification protocols the hybrid system model. However, it is trivial to use it in the share renewal and recovery protocols for synchronous VSS protocols by Herzberg *et al.* [HJKY95]. For both protocols, use of PolyCommit reduces the broadcast size by a linear factor.

For the share renewal protocol, a dealer node has to prove that the constant term of the shared polynomial is zero, which it does by simply publishing the corresponding witness. In the share recovery protocol, a dealer has to prove the evaluation of the polynomial at the index of the recovering node is zero, which again it can do by simply publishing the corresponding witness. Due to their triviality, we do not discuss these modifications in any further detail.

Part II

Applications

Chapter 6

Distributed Private Key Generators for Identity-Based Cryptography

6.1 Preliminaries

In 1984, Shamir [Sha84] introduced the notion of identity-based cryptography (IBC) as an approach to simplify public-key and certificate management in a public-key infrastructure (PKI) and presented an open problem to provide an identity-based encryption (IBE) scheme. After seventeen years, Boneh and Franklin [BF01] proposed the first practical and secure IBE scheme (BF-IBE) using bilinear maps. After this seminal work, in the last few years, significant progress has been made in IBC in the forms of hierarchical IBE schemes, identity-based signature (IBS) schemes, identity-based authentication and key agreement protocols, and other identity-based primitives [JN08].

In an IBC system, a client chooses an arbitrary string such as her e-mail address to be her public key. Consequently, with a standardized public-key string format, an IBC scheme completely eliminates the need for public-key certificates. As an example, in an IBE scheme, a sender can encrypt a message for a receiver knowing just the identity of the receiver and importantly, without obtaining and verifying the receiver's public-key certificate. Naturally, in such a system, a client herself is not capable of generating a private key for her identity. There is a trusted party called a *private-key generator* (PKG) which performs the system setup, generates a secret called the *master key* and provides private keys to clients using it. As the PKG computes a private key for a client, it can decrypt all of her messages passively. This inherent *key escrow* property asks for complete trust in the PKG, which is difficult to find in many realistic scenarios.

Need for a Distributed PKG. Importantly, the amount of trust placed in the holder of an IBC master key is far greater than that placed in the holder of the private key of a certificate authority (CA) in a PKI. In a PKI, in order to attack a client, the CA has to actively generate a fake certificate for the client containing a fake public key. In this case, it is often possible for the client to detect and prove the malicious behaviour of the CA. The CA cannot perform any passive attack; specifically, it cannot decrypt a message encrypted for the client using a client-generated public key and it cannot sign some document for the client, if the verifier gets a correct certificate from the client. On the other hand, in IBC,

- knowing the master key, the PKG can decrypt or sign the messages for any client, without any active attack and consequent detection (key escrow),
- in a key revocation system based on validity periods [BF01], bringing down the PKG is sufficient to bring the system to a complete halt (*single point of failure*), once the current validity period ends.

Therefore, the PKG in IBC needs to be far more trusted than the CA in a PKI. This has been considered as a reason for the slow adoption of IBC schemes outside of closed organizational settings.

Boneh and Franklin [BF01] suggest distributing a PKG in their BF-IBE scheme to solve these problems. In an *(n, t)-distributed PKG*, the master key is distributed among n PKG nodes such that a set of nodes of size t or smaller cannot compute the master key, while a client extracts her private key by obtaining private-key shares from any $t+1$ or more nodes; she can then use the system's public key to verify the correctness of her thus-extracted key. Boneh and Franklin [BF01] propose to design a distributed PKG using VSS of the master key among multiple PKGs and also hint towards a completely distributed approach using the DKG scheme of Gennaro *et al.* [GJKR99]; however, they do not provide a formal model and a security proof. Further, none of the IBE schemes defined after [BF01] consider the design of a distributed PKG. As discussed in Chapter 4, from a practicality standpoint, the DKG schemes [GJKR99] suggested in [BF01] to design a distributed PKG are not advisable for use over the Internet.

As a whole, although various proposed practical applications using IBE, such as key distribution in ad-hoc networks [KKA03], pairing-based onion routing [KZG07] (see Chapter 7) or verifiable random functions from identity-based key encapsulation [ACF09], have a distributed PKG as a fundamental need, there is no distributed PKG available for use over the Internet yet. Defining efficient distributed PKGs for various IBE schemes which can correctly function over the Internet has been an open problem for some time. This practical need forms the motivation of this work.

Related Work. Although we are defining protocols for IBE schemes, as we are concentrating on distributed cryptographic protocols, we do not include a comprehensive account of IBE here. We refer readers to [Boy08] for a detailed discussion on the various IBE schemes and frameworks defined in the literature. Pursuant to this survey, we work in the random oracle model for efficiency and practicality reasons.

None of the IBE schemes except BF-IBE considered distributed PKG setup and key extraction protocols in their design. Recently, Geisler and Smart [GS09] defined a distributed PKG for Sakai and Kasahara's IBE (SK-IBE) [SK03]; however, their solution against a Byzantine (malicious) adversary has an exponential communication complexity and a formal security proof is also not provided. We overcome both of these barriers in our distributed PKG for SK-IBE: our scheme is secure against a Byzantine adversary and has the same polynomial communication complexity as their scheme, which is secure only against an honest-but-curious adversary; we also provide a formal security proof. Other than [GS09], there have been a few other efforts in the literature to counter the inherent key escrow and single point of failure issues in IBE, and next, we compare these alternatives with distributed PKG.

Al-Riyami and Paterson [ARP03] introduce *certificateless cryptography* (CLC) to address the key escrow problem by combining IBC with PKC. Their elegant approach, however, does not address the single point of failure problem. Although it is possible to solve the problem by distributing their PKG using a VSS (which employs a trusted dealer to generate and distribute the key shares), which is inherently cheaper than a DKG-based PKG by a linear factor, it is impossible to stop a dealer's active attacks without completely distributed master-key generation. Further, as private-key extractions are less frequent than encryptions, it is certainly advisable to use more efficient options during encryption rather than private-key extraction. Finally, with the requirement of online access to the receiver's public key, CLC becomes ineffective for systems without continuous network access, where IBC is considered to be an important tool. Lee *et al.* [LBD$^+$04] and Gangishetti *et al.* [GGDS07] propose variants of the distributed PKG involving a more trustworthy key generation centre (KGC) and other key privacy authorities (KPAs). As observed by Chunxiang *et al.* [CJZ05] for [LBD$^+$04], these approaches are, in general, vulnerable to passive attack by the KGC. In addition, the trust guarantees required by a KGC can be unattainable in practice. Goyal [Goy07] reduces the required trust in the PKG by restricting its ability to distribute a client's private key. This does not solve the problem of single point of failure. Further, the PKG in his system still can decrypt the clients' messages passively, which leaves a secure and practical implementation of distributed PKGs wanting.

Threshold versions of signature schemes obtained from some IBE schemes using the Naor transform have been proposed and proved previously [Bol03, WZF05]. However, these solutions do not work for the corresponding IBE scheme. This is due to the inherent secret nature of a client's private keys and corresponding shares as compared to the inherent public nature of signatures and corresponding signature shares. While designing IBE schemes with a distributed PKG, we have to make sure that a PKG node cannot derive more information than the private-key share it generates for a client and that private-key shares are not available in public as commitments.

Contributions. We present distributed PKGs for all three important IBE frameworks: namely, full-domain-hash IBEs, exponent-inversion IBEs and commutative-blinding IBEs [Boy08]. We propose distributed PKG setups and distributed private-key extraction protocols for Boneh and Franklin's IBE (BF-IBE) [BF01], Sakai and Kasahara's IBE (SK-IBE) [SK03], and Boneh and Boyen's (modified) BB$_1$-IBE [Boy08, BM07] schemes for use over the Internet. The novelty of our protocols lies in achieving the secrecy of a client private key from the generating PKG nodes without compromising the efficiency. We realize this with an appropriate use of non-interactive proofs of knowledge, bilinear-pairing-based verifications and DKG protocols with and without the uniform randomness property. Based on the choice of the DKG protocol, our distributed PKGs can work in the synchronous or asynchronous communication model. In terms of feasibility, we ensure that our protocols work for all three bilinear pairing types defined by Galbraith *et al.* [GPS08].

We prove adaptive chosen ciphertext security (IND-ID-CCA) of the defined schemes in the random oracle model. Interestingly, compared to the security proofs for the respective IBE schemes with a single PKG, there are no additional security reduction factors in our proofs, even though the underlying DKG protocol used in the distributed PKGs does not provide a guarantee about the uniform randomness for the generated master secrets. To the best of our knowledge, there is no threshold cryptographic protocol available in the literature where a similar tight security reduction has been proven while using a DKG without the (more expensive) uniform randomness property. Finally, using operation counts, key sizes, and possible pairing types, we compare the performance of three distributed PKGs and also briefly discuss the proactive security and group modification primitives for them.

In Section 6.2, we discuss our assumptions and describe cryptographic tools that we use. With this background, in Section 6.3, we define and prove distributed PKG protocols for the BF-IBE, SK-IBE and BB$_1$-IBE schemes. In Section 6.4, we compare the IBE schemes based on their distributed PKGs and touch upon proactive security and group modification protocols for the system.

6.2 Cryptographic Tools

In this section, we describe important cryptographic tools required to design distributed PKGs in the hybrid model. Note that these tools are the efficient versions of their original forms in [BOGW88, BIB89, GRR98] that utilize the presence of random oracles and the pairing-based DDH problem solving technique [JN03] and are also useful in other asynchronous computational multiparty settings. We start by describing the system and cryptographic assumptions that we make in this chapter.

Assumptions. We follow the hybrid system model of Chapter 4 as it closely depicts the Internet and as the corresponding DKG forms the basis of our distributed PKGs. As a result, our system model has an asynchronous network of $n \geq 3t + 2f + 1$ nodes with a t-limited Byzantine adversary and f-limited crashes and network failures.

As mentioned a number of times above, for efficiency reasons, we use the random oracle framework. Further, our adversary is computationally bounded with a security parameter κ. We assume an instance of a pairing infrastructure of multiplicative groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$, whose common order p is a prime with κ-bit security. For the security of the IBE schemes, we use the BDH and BDHI assumptions, while for commitments and proofs of knowledge, we use the DLog assumption. Refer to Chapter 2 for their definitions.

DKG over $\mathbb{Z}_p$. We use the HybridDKG protocol defined in Section 4.4. For our distributed PKG, we also require the HybridDKG version having the uniform randomness property. Recall their definition in Equations (4.1) through (4.4).

We also need distributed random sharing over $\mathbb{Z}_p$ that generates shares of a secret z chosen jointly at random from $\mathbb{Z}_p$. For the generator $g \in \mathbb{G}$, every node generates a random $r_i \in \mathbb{Z}_p$ and shares that using the HybridDKG-Sh protocol with DLog or Pedersen commitments as $\mathsf{HybridDKG\text{-}Sh}(n, t, f, \tilde{t} = t + 1, g, [h], r_i, [r'_i])$ where the generator $h \in \mathbb{G}$ and randomness r'_i are only required if Pedersen commitments are used. Liveness, agreement, correctness, secrecy and message and communication complexities remain the same as those of the HybridDKG-Sh protocol. We represent the corresponding protocols as follows:

$$\left(\mathcal{C}^{(z)}_{\langle g \rangle}, z_i\right) = \mathsf{Random}_{\mathsf{DLog}}(n, t, f, g) \tag{6.1}$$

$$\left(\mathcal{C}^{(z,z')}_{\langle g,h \rangle}, [\mathcal{C}^{(z)}_{\langle g \rangle}, \mathrm{NIZKPK}_{\mathsf{DLog}}], z_i, z'_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n, t, f, g, h). \tag{6.2}$$

Recall that $\mathcal{C}^{(z)}_{\langle g \rangle} = [g^z, g^{\phi(1)}, \cdots, g^{\phi(n)}]$ and $\mathcal{C}^{(z,z')}_{\langle g,h \rangle} = [g^z h^{z'}, g^{\phi(1)} h^{\phi'(1)}, \cdots, g^{\phi(n)} h^{\phi'(n)}]$ are respectively the DLog and Pedersen commitment vectors for z, where $\phi, \phi' \in \mathbb{Z}_p[x]$ are of degree t with $\phi(0) = z$, $\phi'(0) = z'$, $\phi(i) = z_i$ and $\phi'(i) = z'_i$.

Distributed Addition over $\mathbb{Z}_p$. Let $\alpha, \beta \in \mathbb{Z}_p$ be two secrets shared among n nodes using the HybridDKG-Sh protocol with DLog commitments. Let polynomials $\phi(x), \psi(x) \in$

$\mathbb{Z}_p[x]$ be the respectively associated degree-t polynomials and let $c \in \mathbb{Z}_p$ be a non-zero constant. Due to the linearity of Shamir secret sharing [Sha79], a node P_i with shares α_i and β_i can locally generate shares of $\alpha + \beta$ and $c\alpha$ by computing $\alpha_i + \beta_i$ and $c\alpha_i$, where $\phi(x) + \psi(x)$ and $c\phi(x)$ are the respective polynomials. $\phi(x) + \psi(x)$ is random if either one of $\phi(x)$ or $\psi(x)$ is, and $c\phi(x)$ is random if $\phi(x)$ is. Commitment entries for the resultant shares respectively are $\left(\mathcal{C}^{(\alpha+\beta)}_{\langle g\rangle}\right)_i = \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}\right)_i \left(\mathcal{C}^{(\beta)}_{\langle g\rangle}\right)_i$ and $\left(\mathcal{C}^{(c\alpha)}_{\langle g\rangle}\right)_i = \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}\right)^c_i$. An analogous statement holds when Pedersen commitments are used.

Distributed Multiplication over $\mathbb{Z}_p$. Unlike addition, local distributed multiplication of two shared secrets α and β looks unlikely. We use a distributed multiplication protocol against a computational adversary by Gennaro *et al.* [GRR98, Section 4]. However, instead of their interactive zero-knowledge proof, we utilize the pairing-based DDH problem solving technique [JN03] to verify the correctness of the product value shared by a node non-interactively. For shares α_i and β_i with DLog commitments g^{α_i} and $\hat{g}^{\beta_i}$, given a commitment $g^{\alpha_i\beta_i}$ of the shared product, other nodes can verify its correctness by checking if $e(g^{\alpha_i}, \hat{g}^{\beta_i}) \stackrel{?}{=} e(g^{\alpha_i\beta_i}, \hat{g})$ provided the groups generated by g and $\hat{g}$ are pairing friendly. We observe that it is also possible to perform this verification when one of the involved commitments is a Pedersen commitment. However, if both commitments are Pedersen commitments, then we have to compute DLog commitments for one of the values and employ NIZKPK$_{\mathsf{DLog}}$ to prove its correctness in addition to using the pairing-based verification. In such a case, the choice between the latter technique and the non-interactive version of zero-knowledge proof suggested by Gennaro *et al.* [GRR98] depends upon the implementation efficiencies of the group operation and pairing computations.

In our IBC schemes, we always use the multiplication protocol with at least one DLog commitment. We denote the multiplication protocol involving two DLog commitments as $\mathsf{Mul}_{\mathsf{DLog}}$ and the one involving a combination of the two types of commitments as $\mathsf{Mul}_{\mathsf{Ped}}$.

$$\left(\mathcal{C}^{(\alpha\beta)}_{\langle g^*\rangle}, (\alpha\beta)_i\right) = \mathsf{Mul}_{\mathsf{DLog}}(n, t, f, g^*, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right), \left(\mathcal{C}^{(\beta)}_{\langle \hat{g}\rangle}, \beta_i\right)) \quad (6.3)$$

$$\left(\mathcal{C}^{(\alpha\beta,\alpha\beta')}_{\langle \hat{g},\hat{h}\rangle}, (\alpha\beta)_i, (\alpha\beta')_i\right) = \mathsf{Mul}_{\mathsf{Ped}}(n, t, f, \hat{g}, \hat{h}, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right), \left(\mathcal{C}^{(\beta,\beta')}_{\langle \hat{g},\hat{h}\rangle}, \beta_i, \beta'_i\right)) \quad (6.4)$$

For $\mathsf{Mul}_{\mathsf{DLog}}$, $g^* = g$ or $\hat{g}$. For $\mathsf{Mul}_{\mathsf{Ped}}$, without loss of generality, we assume that β is distributed with the Pedersen commitment. If instead α uses Pedersen commitment, then the Pedersen commitment groups for $(\alpha\beta)$ change to g and h instead of $\hat{g}$ and $\hat{h}$.

Briefly, the protocol works as follows. Every honest node runs the $\mathsf{HybridDKG\text{-}Sh}(n, t, f, \tilde{t} = 2t + 1, \hat{g}, [\hat{h}], \alpha_i\beta_i, [\alpha_i\beta'_i])$ from Eq. 4.1 or 4.3. As discussed above, pairing-based DDH solving is used to verify that the shared value is equal to the product of α_i and β_i.[1] At

[1]For type 3 pairings, a careful selection of commitment generators is required to make the pairing-based verification possible.

the end of the HybridDKG-Sh protocol, instead of adding the subshares of the selected VSS instances, every node interpolates them at index 0 to get the new share $(\alpha\beta)_i$ of $\alpha\beta$.

This protocol is almost equivalent to the share renewal protocol in Section 5.2.1 which is a slight modification of protocol HybridDKG-Sh. The liveness and agreement proofs are exactly the same as those of HybridDKG-Sh. The basic correctness proof remains the same as that of the share renewal protocol except the starting polynomial is of degree $2t+1$ here. On the other hand, the pairing-based DDH problem solving technique assures that the value shared by a node P_i is equal to the product of its shares α_i and β_i. The basic secrecy proof is same as that of the renewal protocol. Further, the adversary cannot determine α or β even after $\alpha\beta$ is reconstructed as the final shared polynomial for $\alpha\beta$ is independent of the shared polynomials for α and β individually. The message and communication complexities are the same as those of the DKG protocol.

As the distributed addition can be performed locally, the above Mul protocols can be seamlessly extended for distributed computation of any expression having binary products (BPs). For ℓ shared secrets $x_1, x_2, \cdots, x_\ell$, and their corresponding DLog commitments $\mathcal{C}^{(x_1)}_{\langle g \rangle}, \mathcal{C}^{(x_2)}_{\langle g \rangle}, \cdots, \mathcal{C}^{(x_\ell)}_{\langle g \rangle}$, shares of any binary product $x' = \sum_{i=1}^{m} k_i x_{a_i} x_{b_i}$ with known constants k_i and indices a_i, b_i can be easily computed by extending the protocol in Eq. 6.3. We denote this generalization as follows:

$$\left(\mathcal{C}^{(x')}_{\langle g^* \rangle}, x'_i\right) \quad = \quad \mathsf{Mul}_{\mathsf{BP}}(n, t, f, g^*, \{(k_i, a_i, b_i)\}, \left(\mathcal{C}^{(x_1)}_{\langle g \rangle}, (x_1)_i\right), \cdots, \left(\mathcal{C}^{(x_\ell)}_{\langle g \rangle}, (x_\ell)_i\right)) \quad (6.5)$$

Node P_j shares $\sum_i k_i (x_{a_i})_j (x_{a_i})_j$. For a type 1 pairing, verification of the correctness of the sharing is done by other nodes as follows.

$$e(g^{\sum_i k_i (x_{a_i})_j (x_{b_i})_j}, g) \stackrel{?}{=} \prod_i e((g^{(x_{a_i})_j})^{k_i}, g^{(x_{b_i})_j})$$

For type 2 and 3 pairings, NIZKPK$_{\equiv DLog}$ is used to provide DLog commitments to the $(x_{b_i})_j$ with generator $\hat{g}$, and then a pairing computation like the above is used. We use the protocol in Eq. 6.5 during distributed private-key extraction in Boneh and Boyen's BB$_1$-IBE scheme in Section 6.3.5.

Sharing the Inverse of a Shared Secret. Given an (n, t, f)-distributed secret α, computing shares of its inverse α^{-1} in distributed manner (without reconstructing α) can be done trivially but inefficiently using a distributed computation of α^{p-2}; this involves $O(\log p)$ distributed multiplications. However, using a technique by Bar-Ilan and Beaver [BIB89], this can be done using just one Random, one Mul and one HybridDKG-Rec protocol. Note that the inverse operation is not possible for $\alpha = 0$; that $\alpha \neq 0$ can easily be verified before beginning the protocol by any party from commitment $\mathcal{C}^{(\alpha)}_{\langle g \rangle}$.

This protocol involves a $\mathsf{HybridDKG\text{-}Rec}$, which outputs the product of the shared secret α with a distributed random element z. If z is created using DLog commitments and is not uniformly random, the product αz may leak some information about α. We avoid this by using Pedersen commitments while generating z. We represent this protocol as follows:

$$\left(\mathcal{C}^{(\alpha^{-1})}_{\langle g^*\rangle}, (\alpha^{-1})_i\right) \quad = \quad \mathsf{Inverse}(n, t, f, \hat{g}, \hat{h}, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right)) \tag{6.6}$$

Here g^* belongs to any group of order p. The liveness, agreement and secrecy properties of the protocol are the same as those of $\mathsf{HybridDKG\text{-}Sh}$ except secrecy is defined in the terms of α^{-1} instead of α; for the correctness property, along with recoverability to a unique value s, this protocol additionally mandates that $s = \alpha^{-1}$. For a distributed secret $\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right)$, protocol $\mathsf{Inverse}$ works as follows: every node P_i:

1. runs $\left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h}\rangle}, z_i, z'_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n, t, f, \hat{g}, \hat{h})$;

2. computes shares of $(w, w') = (\alpha z, \alpha z')$ as $\left(\mathcal{C}^{(w,w')}_{\langle \hat{g},\hat{h}\rangle}, w_i, w'_i\right) = \mathsf{Mul}_{\mathsf{Ped}}(n, t, f, \hat{g}, \hat{h},$ $\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right), \left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h}\rangle}, z_i, z'_i\right))$;

3. sends (w_i, w'_i) to each node and reconstructs $w = \mathsf{HybridDKG\text{-}Rec}_{\mathsf{Ped}}(t, \mathcal{C}^{(w,w')}_{\langle \hat{g},\hat{h}\rangle}, w_i, w'_i)$. If $w = 0$, which happens with negligible probability (when $z = 0$), repeats the above two steps, else locally computes $(\alpha^{-1})_i = w^{-1} z_i$;

4. computes the commitment $\mathcal{C}^{(\alpha^{-1})}_{\langle g^*\rangle}$ using w^{-1}, $\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h}\rangle}$, and if required, any of the NIZKPK techniques.

A modified form of this protocol is used in Section 6.3.4.

This protocol is a combination of the $\mathsf{Random}_{\mathsf{Ped}}$, $\mathsf{Mul}_{\mathsf{Ped}}$ and $\mathsf{HybridDKG\text{-}Rec}$ protocols along with some local computations. Therefore, its liveness and agreement properties follow directly from the corresponding properties of protocol $\mathsf{HybridDKG}$. Uniqueness of the recovered value follows from the correctness property of protocol $\mathsf{HybridDKG}$, while its equality to α^{-1} can be proven as follows: a share computed by a node P_i at the end of protocol $\mathsf{Inverse}$ is equal to $\frac{z_i}{z\alpha}$, where $\mathcal{C}^{(\frac{z}{z\alpha})}_{\langle g^*\rangle}$ is the associated commitment vector. When reconstructed, it provides α^{-1} as follows:

$$\mathsf{HybridDKG\text{-}Rec}_{\mathsf{DLog}}(t, \mathcal{C}^{(\frac{z}{z\alpha})}_{\langle g\rangle}, \frac{z_i}{z\alpha}) = \frac{1}{z\alpha}\mathsf{HybridDKG\text{-}Rec}_{\mathsf{DLog}}(t, \mathcal{C}^{(z)}_{\langle g\rangle}, z_i) = \frac{z}{z\alpha} = \alpha^{-1}$$

Secrecy of protocol $\mathsf{Inverse}$ follows directly from secrecy of protocols $\mathsf{HybridDKG\text{-}Sh}_{\mathsf{Ped}}$ and Mul. After the reconstruction of $w = z\alpha$, the distributed uniformly random element z and

α remain private by the secrecy properties of protocol Mul. As the final shares of α^{-1} are generated using a local computation, there is no secrecy loss in the last step either. It has the same asymptotic message and communication complexities as those of protocol HybridDKG-Sh.

6.3 Distributed Private Key Generators (PKGs) for Identity-Based Encryption (IBE) Schemes

We present and prove distributed PKG setup and private key extraction protocols for three IBE schemes: namely, Boneh and Franklin's IBE (BF-IBE) [BF01], Sakai and Kasahara's IBE (SK-IBE) [SK03], and Boneh and Boyen's IBE (BB_1-IBE) [Boy08]. Each of these schemes represents a distinct important category of an IBE classification defined by Boyen [Boy07]. They respectively belong to *full-domain-hash* IBE schemes, *exponent-inversion* IBE schemes, and *commutative-blinding* IBE schemes. Note that the distributed PKG architectures that we develop for each of the three schemes apply to every scheme in their respective categories. Our above choice of IBE schemes is influenced by an identity-based cryptography standard (IBCS) [BM07] and also a comparative study by Boyen [Boy08], which finds the above three schemes to be the most practical IBE schemes in their respective categories. In his classification, Boyen [Boy07] also includes another category for quadratic-residuosity-based IBE schemes; however, none of the known schemes in this category are practical enough to consider here.

The role of a PKG in an IBE scheme ends with a user's private-key extraction. The distributed form of the PKG does not affect the encryption and decryption steps of IBE. Consequently, we concentrate only on the distributed PKG setup and private-key extraction steps of the three IBE schemes under consideration. However, we recall the original encryption and decryption definitions for our proofs. We start by describing a bootstrapping procedure required by all IBE schemes.

6.3.1 Bootstrapping Procedure

Each of the IBE schemes under consideration here requires the following three bootstrapping steps.

1. Determine the node group size n, the security threshold t and the crashed-nodes threshold f such that $n \geq 3t + 2f + 1$.
2. Choose the pairing type to be used and select three groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ of prime order p such that there exists a bilinear pairing $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ of the decided type. The group order p is determined by the security parameter κ.

3. Choose two generators $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$ required to generate public parameters as well as the commitments. With a type 1 or 2 pairing, set $g = \varphi(\hat{g})$.

Any untrusted entity can perform these offline tasks. Honest DKG nodes can verify the correctness of the tuple (n, t, f) and confirm the group choices $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ as the first step of their distributed PKG setup. If unsatisfied, they may decline to proceed. We denote the generated bilinear pairing group as $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_t \rangle$.

6.3.2 Formal Security Model

An IBE scheme with an (n, t, f)-distributed PKG consists of the following four components:

- A *distributed PKG setup protocol* for node P_i that takes the above bootstrapped parameters n, t, f, and $\mathcal{G}$ as input and outputs a share s_i of a shared master secret s and a corresponding public-key vector K_{pub} of a master public key and n public-key shares.
- A *distributed private key-extraction protocol* for node P_i that takes a client identity `ID`, the public key vector K_{pub} and the master-secret share s_i as input and outputs a verifiable private-key share $(d_{\mathtt{ID}})_i$. The client computes the private key $d_{\mathtt{ID}}$ after verifying the received shares $(d_{\mathtt{ID}})_i$.
- An *encryption algorithm* that takes a receiver identity `ID`, the public key vector K_{pub} (specifically, the master public key) and a plaintext message M as input and outputs a ciphertext C.
- A *decryption algorithm* for client with identity `ID` that takes a ciphertext C and the private key $d_{\mathtt{ID}}$ as input and outputs a plaintext M.

Note that the above distributed PKG setup protocol does not require any *dealer* and that we mandate verifiability for the private-key shares rather than obtaining robustness using error-correcting techniques. During private-key extractions, we insist on minimal interaction between clients and PKG nodes—transferring identity credentials from the client at the start and private-key shares from the nodes at the end.

To define security against an IND-ID-CCA attack, we consider the following game that a challenger plays against a polynomially bounded t-limited Byzantine adversary with f-limited crashes and link failures.

Setup. The adversary chooses to corrupt a fixed set of t nodes. To run a distributed PKG setup protocol, the challenger simulates the remaining $n - t$ nodes. Of these, the adversary can further crash any f nodes at any instant. Modelling these f crashed nodes is trivial. The adversary informs the indices of the crashed nodes to the challenger, who makes sure not to use the inputs corresponding to those f nodes during the period they are crashed. It, however, computes the internal states of the crashed nodes using the outputs corresponding to other $n-t-f$ nodes that it runs. When the adversary modifies its choice of crashed nodes, the challenger models the associated recoveries using the internal states computed during the protocol. Note that, for the simplicity and clarity of the protocols and the proofs, we ignore these f crashes in exposition of our distributed PKG setup and private-key extraction protocols.

At the end of the protocol execution, the adversary receives t shares of a shared master secret for its t nodes and a public key vector K_{pub}. The challenger knows the remaining $n-t$ shares and can derive the master secret as $n-t-f \geq t+1$.

Phase 1. The adversary adaptively issues private-key extraction and decryption queries to the challenger. For a private-key extraction query $\langle \mathtt{ID} \rangle$, the challenger runs the distributed key extraction protocol for its $n-t$ nodes, interacts with the t adversary nodes, and sends verifiable private-key shares for its $n-t-f$ nodes. For a decryption query $\langle \mathtt{ID}, C \rangle$, the challenger decrypts C by generating the private key $d_{\mathtt{ID}}$ or using the master secret.

Challenger. The adversary chooses two equal-length plaintexts M_0 and M_1, and a challenge identity $\mathtt{ID}_{ch}$ such that $\mathtt{ID}_{ch}$ does not appear in any private-key extraction query in Phase 1. The challenger chooses $b \in_R \{0,1\}$ and encrypts M_b for $\mathtt{ID}_{ch}$, and gives the ciphertext C_{ch} to the adversary.

Phase 2. The adversary adaptively issues more private-key extraction and decryption queries to the challenger except for a key extraction query for $\langle \mathtt{ID}_{ch} \rangle$ and decryption queries for $\langle \mathtt{ID}_{ch}, C_{ch} \rangle$.

Guess. Finally, the adversary outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

Security against IND-ID-CCA attacks means that, for any polynomially bounded adversary, $b' = b$ with probability negligibly greater than 1/2.

6.3.3 Boneh and Franklin's BF-IBE

BF-IBE [BF01] belongs to the full-domain-hash IBE family. In a BF-IBE setup, a PKG generates a master key $s \in \mathbb{Z}_p$ and an associated public key $g^s \in \mathbb{G}$, and derives private keys ($d \in \hat{\mathbb{G}}$) for clients using their well-known identities and s. A client with identity $\mathtt{ID}$ receives the private key $d_{\mathtt{ID}} = \left(\hat{H}(\mathtt{ID})\right)^s = h_{\mathtt{ID}}^s \in \hat{\mathbb{G}}$, where $\hat{H} : \{0,1\}^* \to \hat{\mathbb{G}}^*$ is a full-domain cryptographic hash function. ($\hat{\mathbb{G}}^*$ denotes the set of all elements in $\hat{\mathbb{G}}$ except the identity.) The security of BF-IBE is based on the BDH assumption.

Distributed PKG Setup. The distributed PKG setup involves the generation of the system master key and the associated system public-key tuple in the (n, t)-distributed form among n nodes. Each node P_i participates in a common DKG over $\mathbb{Z}_p$ to generate its share $s_i \in \mathbb{Z}_p$ of the distributed master key s. The system public-key tuple is of the form $\mathcal{C}^{(s)}_{\langle g \rangle} = [g^s, g^{s_1}, \cdots, g^{s_n}]$. We obtain this using our $\mathsf{Random}_{\mathsf{DLog}}$ protocol from Eq. 6.1 as

$$\left(\mathcal{C}^{(s)}_{\langle g \rangle}, s_i\right) \quad = \quad \mathsf{Random}_{\mathsf{DLog}}(n, t, g)$$

Private-key Extraction. As a client needs $t + 1$ correct shares, it is sufficient for the client to contact any $2t + 1$ nodes (say set $\mathcal{Q}$). The private-key extraction protocol works as follows.

1. Once a client with identity $\mathtt{ID}$ contacts every node in $\mathcal{Q}$, every honest node $P_i \in \mathcal{Q}$ authenticates the client's identity and returns a private-key share $h^{s_i}_{\mathtt{ID}} \in \hat{\mathbb{G}}$ over a secure and authenticated channel.

2. Upon receiving $t + 1$ valid shares, the client can construct her private key $d_{\mathtt{ID}}$ as $d_{\mathtt{ID}} = \prod_{P_i \in \mathcal{Q}} (h^{s_i}_{\mathtt{ID}})^{\lambda_i} \in \hat{\mathbb{G}}$, where the Lagrange coefficient $\lambda_i = \prod_{P_j \in \mathcal{Q} \setminus \{i\}} \frac{j}{j-i}$.

3. The client can verify the correctness of the computed private key $d_{\mathtt{ID}}$ by checking $e(g, d_{\mathtt{ID}}) \stackrel{?}{=} e(g^s, h_{\mathtt{ID}})$. If unsuccessful, she can verify the correctness of each received $h^{s_i}_{\mathtt{ID}}$ by checking if $e(g, h^{s_i}_{\mathtt{ID}}) \stackrel{?}{=} e(g^{s_i}, h_{\mathtt{ID}})$. An equality proves the correctness of the share, while an inequality indicates misbehaviour by the node P_i and its consequential removal from $\mathcal{Q}$.

In asymmetric pairings, elements of $\mathbb{G}$ generally have a shorter representation than those of $\hat{\mathbb{G}}$. Therefore, we put the more frequently accessed system public-key shares in $\mathbb{G}$, while the occasionally transferred client private-key shares belong to $\hat{\mathbb{G}}$. This also leads to a reduction in the ciphertext size. However, for type 2 pairings, an efficient hash-to-$\hat{\mathbb{G}}$ is not available for the group $\hat{\mathbb{G}}$ [GPS08]; in that case we compute the system public key shares in $\hat{\mathbb{G}}$ and use the more feasible group $\mathbb{G}$ for the private key shares.

Encryption and Decryption. Boneh and Franklin obtain an IND-ID-CCA secure IBE encryption protocol ($\mathsf{FullIdent}$) [BF01, Section 4.2] secure against the BDH assumption by applying the Fujisaki-Okamoto transformation [FO99] to their IND-ID-CPA secure scheme ($\mathsf{BasicIdent}$). Along with $\hat{H} : \{0,1\}^* \to \hat{\mathbb{G}}^*$, this scheme uses three more random oracles: $H_2 : \mathbb{G}_t \to \{0,1\}^{\kappa'}$, $H_3 : \{0,1\}^{\kappa'} \times \{0,1\}^{\kappa'} \to \mathbb{Z}_p$, and $H_4 : \{0,1\}^{\kappa'} \to \{0,1\}^{\kappa'}$. Here κ' is a security parameter that must be at least 2κ.

Encryption: To encrypt a message M of fixed bit length κ' for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^{\kappa'}$, computes $r = H_3(\sigma, M)$ and $h_{\mathtt{ID}} = \hat{H}(\mathtt{ID})$, and sends $C = (u, v, w) = (g^r, \sigma \oplus H_2(e(g^s, h_{\mathtt{ID}})^r), M \oplus H_4(\sigma))$ to the receiver.

Decryption: To decrypt a ciphertext $C = (u, v, w)$ using the private key $d_{\mathtt{ID}}$, the receiver successively computes $\sigma = v \oplus H_2(e(u, d_{\mathtt{ID}}))$, $M = w \oplus H_4(\sigma)$, and $r = H_3(\sigma, M)$. If $g^r \neq u$, then the receiver rejects C, else it accepts M as a valid message.

Proof of Security. We prove the IND-ID-CCA security of BF-IBE with the (n,t)-distributed PKG ((n,t)-$\mathsf{FullIdent}$) based on the BDH assumption in the random oracle model. Hereafter, q_E, q_D and q_{H_i} denote the number of extraction, decryption and random oracle H_i queries respectively.

Theorem 6.1. *Let $\hat{H}$, H_2, H_3 and H_4 be random oracles. Let $\mathcal{A}_1$ be an IND-ID-CCA adversary that has advantage $\epsilon_1(\kappa)$ in running time $t_1(\kappa)$ against (n,t)-$\mathsf{FullIdent}$ making at most q_E, q_D, $q_{\hat{H}}$, q_{H_2}, q_{H_3}, and q_{H_4} queries. Then, there exists an algorithm $\mathcal{B}$ that solves the BDH problem in $\mathcal{G}$ with advantage roughly equal to $\epsilon_1(\kappa)/(q_{\hat{H}} q_{H_2}(q_{H_3} + q_{H_4}))$ and running time $O(t_1(\kappa), q_E, q_D, q_{\hat{H}}, q_{H_2}, q_{H_3}, q_{H_4})$.*

For their proof, Boneh and Franklin define two additional public key encryption schemes: IND-CPA secure $\mathsf{BFBasicPub}$ [BF01], and its IND-CCA secure version $\mathsf{BFBasicPub}^{hy}$ [BF01]. We use distributed versions of these schemes: (n,t)-$\mathsf{BFBasicPub}^{hy}$ and (n,t)-$\mathsf{BFBasicPub}$ respectively. Both (n,t)-$\mathsf{BFBasicPub}^{hy}$ and (n,t)-$\mathsf{BFBasicPub}$ protocols have three steps: keygen, $\mathsf{encrypt}$ and $\mathsf{decrypt}$. We first define the protocol (n,t)-$\mathsf{BFBasicPub}$:

keygen: Given a bilinear group $\mathcal{G}$ for a security parameter κ, a set of n nodes runs the BF-IBE distributed PKG setup for threshold t ($n \geq 3t+1$) to generate individual private keys s_i and a public key tuple $\mathcal{C}^{(s)}_{\langle g \rangle}$. n nodes also run protocol $\mathsf{HybridDKG\text{-}Sh}$ to generate $\hat{h}_{\mathtt{ID}} \in_R \hat{\mathbb{G}}$. Assuming a random oracle $H_2 : \mathbb{G} \to \{0,1\}^{\kappa'}$, where κ' is the message length, the system public key is $\langle \mathcal{G}, g, \hat{g}, \mathcal{C}^{(s)}_{\langle g \rangle}, \hat{h}_{\mathtt{ID}}, H_2 \rangle$. Every node generates its private-key share $(d_{\mathtt{ID}})_i = \hat{h}_{\mathtt{ID}}^{s_i}$ corresponding to the system's private key $d_{\mathtt{ID}}$.

$\mathsf{encrypt}$: To encrypt $M \in \{0,1\}^{\kappa'}$, choose $r \in_R \mathbb{Z}_p^*$ and set the ciphertext $C = (g^r, M \oplus H_2(e(g^s, h_{\mathtt{ID}})^r))$.

$\mathsf{decrypt}$: To decrypt the ciphertext $C = (u, v)$ using the private key shares $(d_{\mathtt{ID}})_i$, compute and share $e(u, (d_{\mathtt{ID}})_i)$ with every other node or with a common accumulator. Interpolate these pairing values to generate $e(u, d_{\mathtt{ID}})$ and compute $M = v \oplus H_2(e(u, d_{\mathtt{ID}}))$.

Protocol (n,t)-$\mathsf{BFBasicPub}^{hy}$ only modifies the $\mathsf{encrypt}$ and $\mathsf{decrypt}$ steps of the above protocol using the Fujisaki-Okamoto transformation [FO99], and random oracles $H_3 : \{0,1\}^{\kappa'} \times \{0,1\}^{\kappa'} \to \mathbb{Z}_p$ and $H_4 : \{0,1\}^{\kappa'} \to \{0,1\}^{\kappa'}$.

Boneh and Franklin prove the security of $\mathsf{FullIdent}$ in the following proof sequence: $\mathsf{FullIdent} \rightarrow \mathsf{BFBasicPub}^{hy} \rightarrow \mathsf{BFBasicPub} \rightarrow \mathsf{BDH}$. Galindo [Gal05] corrects a flaw in their proof maintaining the same proof sequence. We also follow the same proof sequence through Lemmas 6.2, 6.3 and 6.4 to prove Theorem 6.1:

$$(n,t)\text{-}\mathsf{FullIdent} \rightarrow (n,t)\text{-}\mathsf{BFBasicPub}^{hy} \rightarrow (n,t)\text{-}\mathsf{BFBasicPub} \rightarrow \mathsf{BDH}.$$

Lemma 6.2. *Let $\hat{H}$, H_2, H_3 and H_4 be random oracles. Let $\mathcal{A}_1$ be an IND-ID-CCA adversary that has advantage $\epsilon(\kappa)$ in running time $t(\kappa)$ against (n,t)-FullIdent. Suppose $\mathcal{A}_1$ makes at most q_E, q_D, $q_{\hat{H}}$, q_{H_2}, q_{H_3}, and q_{H_4} queries. Then there exists an IND-CCA adversary $\mathcal{A}_2$ that has advantage at least $\epsilon(\kappa)/q_{\hat{H}}$ against (n,t)-BFBasicPub^{hy}. Its running time is at most $t(\kappa) + c(nq_E + q_D + q_{\hat{H}})$ where c is the average time of exponentiation in $\hat{\mathbb{G}}$.*

Proof. (Outline) The game between the challenger and the adversary $\mathcal{A}_2$ starts with the challenger running the keygen step of (n,t)-$\mathsf{BFBasicPub}^{hy}$. $\mathcal{A}_2$ simultaneously starts adversary $\mathcal{A}_1$ and forwards all messages from the challenger to $\mathcal{A}_1$ and vice versa. As a result, in this simulation game, t out of n nodes are run by $\mathcal{A}_1$, while the challenger runs the remaining $n-t$ nodes. $\mathcal{A}_2$, however, knows all information gathered by $\mathcal{A}_1$. At the end of the distributed PKG setup, along with $\mathcal{A}_1$'s public parameters, $\mathcal{A}_2$ also knows secret shares s_i for the t nodes run by $\mathcal{A}_1$. The rest of the game and the analysis remains the same as that of [Gal05], except during key extraction queries. Here, instead of a private key $d_{\mathtt{ID}}$, $\mathcal{A}_2$ has to provide $t+1$ private-key shares to $\mathcal{A}_1$. This is, however, easily possible knowing $\mathcal{A}_1$'s t secret shares and the randomness ν_i used in the $\hat{H}$ random oracle hash table $\langle \mathtt{ID}_i, h_i, \nu_i, bit_i\rangle \in \{0,1\}^* \times \hat{\mathbb{G}}^* \times \mathbb{Z}_p^* \times \{0,1\}$. Refer to [Gal05, Section 3] for the rest of the proof. □

Lemma 6.3 (Fujisaki-Okamoto [FO99])**.** *Let H_3 and H_4 be random oracles. Let $\mathcal{A}_2$ be an IND-CCA adversary that has advantage $\epsilon_2(\kappa)$ against (n,t)-BFBasicPub^{hy} making at most q_D, q_{H_3}, and q_{H_4} queries in running time $t_2(\kappa)$. Then there exists an IND-CPA adversary $\mathcal{A}_3$ that has advantage at least $\frac{1}{2(q_{H_3}+q_{H_4})}[(\epsilon_2(\kappa)+1)(1-2/p)^{q_D}-1]$ against (n,t)-BFBasicPub. Its running time is at most $t_2(\kappa) + O((q_{H_3}+q_{H_4})\kappa')$, where κ' is the message length.*

Lemma 6.4. *Let H_2 be a random oracle. Let $\mathcal{A}_3$ be an IND-CPA adversary that has advantage $\epsilon_3(\kappa)$ in running time $t_3(\kappa)$ against (n,t)-BFBasicPub making at most q_{H_2} queries. Then there exists an algorithm $\mathcal{B}$ that solves the BDH problem in $\langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_t\rangle$ with advantage at least $2\epsilon_3(\kappa)/q_{H_2}$ and a running time $O(t_3(\kappa))$.*

Proof. Algorithm $\mathcal{B}$ is given a random instance of the BDH problem $\langle g, \hat{g}, g^a, \hat{g}^a, g^b, \hat{g}^c\rangle$ in a bilinear group $\mathcal{G}$. Let $D = e(g,\hat{g})^{abc} \in \mathbb{G}_t$ be the solution to this problem. Algorithm $\mathcal{B}$ finds D by interacting with $\mathcal{A}_3$ as follows:

Setup: $\mathcal{B}$ runs the keygen step of (n,t)-BFBasicPub using the BDH instance. Let P_{Bad} be the set of t parties corrupted or owned by $\mathcal{A}_3$. Let P_{Good} be the set of remaining good parties which will be run by $\mathcal{B}$. $\mathcal{B}$ wants to make sure that the challenge g^a and $\hat{g}^c$ are included respectively in $g^s \in \mathcal{C}^{(s)}_{\langle g \rangle}$ and $\hat{h}_{\mathtt{ID}}$ of (n,t)-BFBasicPub. As in protocol HybridDKG-Sh, the VSSs selection may not be under $\mathcal{B}$'s control, $\mathcal{B}$ cannot "hit" g^a as the system public key corresponding to s. It rather uses $(g^a)^{\mu_i}$ and $(\hat{g}^c)^{\mu'_i}$ for $\mu_i, \mu'_i \in_R \mathbb{Z}_p^*$ as its contributions towards respectively s and $\hat{h}_{\mathtt{ID}}$ in keygen for every $P_i \in P_{Good}$. More specifically, for every $P_i \in P_{Good}$, $\mathcal{B}$ chooses $\mu_i, \mu'_i \in_R \mathbb{Z}_p^*$ and $s_{ij}, s'_{ij} \in_R \mathbb{Z}_p$ for every $P_j \in P_{Bad}$, where $s_{i,j}$ and s'_{ij} are subshares for P_j of VSSs run by P_i. Although $\mathcal{B}$ does not know the contributions $\mu_i a$ and $\mu'_i c$, it can provide consistent commitment vectors $\mathcal{C}^{(\mu_i a)}_{\langle g \rangle}$ and $\mathcal{C}^{(\mu'_i c)}_{\langle \hat{g} \rangle}$ to $\mathcal{A}_3$ knowing s_{ij}, s'_{ij} for $P_j \in P_{Bad}$, μ_i, μ'_i, g^a, and $\hat{g}^c$. For VSSs run by the adversary nodes $P_j \in P_{Bad}$, $\mathcal{B}$ can reconstruct the exact contributions ν_j and ν'_j using $n-t$ subshares obtained from P_j. Therefore, for any subset of VSSs Q and Q' chosen finally, $s = a\sum_{P_i \in Q_{Good}} \mu_i + \sum_{P_j \in Q_{Bad}} \nu_i$ and $\hat{h}_{\mathtt{ID}} = \hat{g}^{c\sum_{P_i \in Q'_{Good}} \mu'_i + \sum_{P_j \in Q'_{Bad}} \nu'_i}$. Note that B knows $\nu = \sum_{P_j \in Q_{Bad}} \nu_i$, $\nu' = \sum_{P_j \in Q'_{Bad}} \nu'_i$ $\mu = \sum_{P_i \in Q_{Good}} \mu_i$ and $\mu' = \sum_{P_i \in Q'_{Good}} \mu'_i$. Let s_i be the final share of s for each node P_i. Observe that the (unknown) associated private key $d_{\mathtt{ID}} = \hat{g}^{(a\mu+\nu)(c\mu'+\nu')} = \hat{g}^{\mu\mu'(ac)+\mu\nu'(a)+\mu'\nu(c)+\nu\nu'}$. $\mathcal{B}$ runs random oracle H_2 for $\mathcal{A}_3$ creating a list H_2^{list} of $\langle \mathbb{G}_t, \{0,1\}^{\kappa'} \rangle$. An entry $\langle x_i, h_i \rangle$ indicates that $h_i = H_2(x_i)$. Finally, it is easy to see that this simulated view of $\mathcal{A}_3$ is identically distributed as in a real execution of keygen.

The rest of the game and the analysis remains the same as that of [BF01, Lemma 4.3], except during the **Guess** step. Here, instead of returning x_i from a random tuple $\langle x_i, h_i \rangle$ from H_2^{list} as answer to the BDH problem, $\mathcal{B}$ returns

$$\left(\frac{x_i}{e(g^b, \hat{g}^a)^{\mu\nu'} e(g^b, \hat{g}^c)^{\mu'\nu} e(g^b, \hat{g})^{\nu\nu'}} \right)^{(\mu\mu')^{-1}}.$$

□

Here, if x_i is the correct choice, then x_i is equal to $e(g, \hat{g})^{abc\mu\mu' + ab\mu\nu' + bc\mu'\nu + b\nu\nu'}$ instead of $e(g, \hat{g})^{abc}$ in the original BF-IBE proof.

6.3.4 Sakai and Kasahara's SK-IBE

SK-IBE [SK03] belongs to the exponent-inversion IBE family. The PKG setup here remains exactly same as BF-IBE and the PKG generates a master key $s \in \mathbb{Z}_p$ and an associated public key $g^s \in \mathbb{G}$ just as in BF-IBE. However, the key extraction differs significantly. Here, a client with identity ID receives the private key $d_{\mathtt{ID}} = \hat{g}^{\frac{1}{s+H_1(\mathtt{ID})}} \in \hat{\mathbb{G}}$, where H_1 :

$\{0,1\}^* \to \mathbb{Z}_p$. Chen and Cheng [CC05] prove the security of SK-IBE based on the BDHI assumption.

Distributed PKG Setup. The distributed PKG setup remains the exactly same as that of BF-IBE, where $s_i \in \mathbb{Z}_p$ is the master-key share for node P_i and $\mathcal{C}^{(s)}_{\langle g \rangle} = [g^s, g^{s_1}, \cdots, g^{s_n}]$ is the system public-key tuple.

Private-key Extraction. The private-key extraction for SK-IBE is not as straightforward as that for BF-IBE. We modify the $\mathsf{Inverse}$ protocol described in Section 6.2; specifically, here a private-key extracting client receives w_i from the node in step 3 and instead of PKG nodes, the *client* performs the interpolation step of $\mathsf{HybridDKG\text{-}Rec}$. In step 4, instead of publishing, PKG nodes forward $\hat{g}^{z_i}$ and the associated $\text{NIZKPK}_{\mathsf{DLog}}$ directly to the client, which computes $\hat{g}^z$ and then $d_{\texttt{ID}} = (\hat{g}^z)^{w^{-1}}$. The reason behind this is to avoid possible key escrow if the node computes both $\hat{g}^z$ and w. Further, the nodes precompute another generator $\hat{h} \in \hat{\mathbb{G}}$ for Pedersen commitments using $\left(\mathcal{C}^{(r)}_{\langle \hat{g} \rangle}, r_i\right) = \mathsf{Random}_{\mathsf{DLog}}(n, t, \hat{g})$, set $\hat{h} = \left(\mathcal{C}^{(r)}_{\langle \hat{g} \rangle}\right)_0 = \hat{g}^r$, and discard their shares r_i.

1. Once a client with identity $\texttt{ID}$ contacts all n nodes the system, every node P_i authenticates the client's identity, runs $\left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h} \rangle}, z_i, z'_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n, t, \hat{g}, \hat{h})$ and computes $s^{\texttt{ID}}_i = s_i + H_1(\texttt{ID})$ and for $0 \le j \le n$, $\left(\mathcal{C}^{(s^{\texttt{ID}})}_{\langle g \rangle}\right)_j = \left(\mathcal{C}^{(s)}_{\langle g \rangle}\right)_j g^{H_1(\texttt{ID})} = g^{s_j + H_1(\texttt{ID})}$. If $\left(\mathcal{C}^{(s^{\texttt{ID}})}_{\langle g \rangle}\right)_0 = 1$, which happens with negligible probability, abort the protocol because $s^{\texttt{ID}} = 0$.

2. Node P_i performs $\left(\mathcal{C}^{(w,w')}_{\langle \hat{g},\hat{h} \rangle}, w_i, w'_i\right) = \mathsf{Mul}_{\mathsf{Ped}}(n, t, \hat{g}, \hat{h}, \left(\mathcal{C}^{(s^{\texttt{ID}})}_{\langle g \rangle}, s^{\texttt{ID}}_i\right), \left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h} \rangle}, z_i, z'_i\right))$, where $w = s^{\texttt{ID}} z = (s + H_1(\texttt{ID}))z$ and $w' = (s + H_1(\texttt{ID}))z'$ and sends $\left(\mathcal{C}^{(w,w')}_{\langle \hat{g},\hat{h} \rangle}, w_i\right)$ along with $\text{NIZKPK}_{\mathsf{DLog}}(w_i, w'_i, \left(\mathcal{C}^{(w)}_{\langle \hat{g} \rangle}\right)_i, \left(\mathcal{C}^{(w,w')}_{\langle \hat{g},\hat{h} \rangle}\right)_i)$ to the client, which upon receiving $t+1$ verifiably correct shares (w_i) reconstructs w using Lagrange interpolation. If $w \neq 0$, then it computes w^{-1} or else starts again from step 1.

3. Node P_i sends $\left(\mathcal{C}^{(z)}_{\langle \hat{g} \rangle}\right)_i = \hat{g}^{z_i}$ and $\left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h} \rangle}\right)$ along with $\text{NIZKPK}_{\mathsf{DLog}}(z_i, z'_i, \left(\mathcal{C}^{(z)}_{\langle \hat{g} \rangle}\right)_i, \left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h} \rangle}\right)_i)$ to the client.

4. The client verifies $\left(\mathcal{C}^{(z)}_{\langle \hat{g} \rangle}\right)_i$ using the received $\text{NIZKPK}_{\mathsf{DLog}}$, Lagrange-interpolates $t+1$ valid $\hat{g}^{z_i}$ to compute $\hat{g}^z$ and derives her private key $(\hat{g}^z)^{w^{-1}} = \hat{g}^{\frac{1}{(s+H(\texttt{ID}))}}$.

This protocol can be used without any modification with any type of pairing. Further, online execution of the $\mathsf{Random}_{\mathsf{Ped}}$ computation can be eliminated using batch precomputation of distributed random elements $\left(\mathcal{C}^{(z,z')}_{\langle \hat{g},\hat{h}\rangle}, z_i, z'_i\right)$.

Encryption and Decryption. Chen and Cheng [CC05] define an IND-ID-CCA secure version of the SK-IBE scheme secure against the BDHI assumption. Here, the random oracle $\hat{H}$ in BF-IBE is replaced by $H_1 : \{0,1\}^* \to \mathbb{Z}_p$. The other random oracles H_2, H_3 and H_4 remain the same. This scheme also uses Fujisaki-Okamoto transformation [FO99] to achieve IND-ID-CCA security.

Encryption: To encrypt a message M of some fixed bit length κ' for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^{\kappa'}$, computes $r = H_3(\sigma, M)$ and $h_{\mathtt{ID}} = H_1(\mathtt{ID})$, and sends $C = (u, v, w) = ((g^s g^{h_{\mathtt{ID}}})^r, \sigma \oplus H_2(e(g, \hat{g})^r), M \oplus H_4(\sigma))$ to the receiver.

Decryption: To decrypt a ciphertext $C = (u, v, w)$ using the private key $d_{\mathtt{ID}}$, the receiver successively computes $\sigma = v \oplus H_2(e(u, d_{\mathtt{ID}}))$, $M = w \oplus H_4(\sigma)$, and $r = H_3(\sigma, M)$. If $(g^s g^{h_{\mathtt{ID}}})^r \neq u$, then the receiver rejects C, else it accepts M as a valid message.

Proof of Security. The security of SK-IBE with a distributed PKG ((n,t)-SK-IBE) is based on the BDHI assumption.

Theorem 6.5. *Let H, H_1, H_2, H_3 and H_4 be random oracles. Let $\mathcal{A}_1$ be an IND-ID-CCA adversary that has advantage $\epsilon_1(\kappa)$ in running time $t_1(\kappa)$ against (n,t)-SK-IBE making at most q_E, q_D, q_{H_1}, q_{H_2}, q_{H_3}, and q_{H_4} queries. Then, there exists an algorithm $\mathcal{B}$ that solves the* BDHI *problem in $\mathcal{G}$ with advantage roughly equal to $\epsilon_1(\kappa)/(q_{H_1} q_{H_2}(q_{H_3} + q_{H_4}))$ and running time $O(t_1(\kappa), q_E, q_D, q_H, q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4})$.*

Chen and Cheng use the same technique as that of BF-IBE (with the modification by Galindo) to obtain the proof sequence SK-IBE $\to$ $\mathsf{SKBasicPub}^{hy}$ $\to$ $\mathsf{SKBasicPub}$ $\to$ BDHI. We also use the same proof sequence. Here, however, we divert from the proof of Theorem 6.1 for (n,t)-$\mathsf{FullIdent}$. To prove Theorem 6.5 for (n,t)-SK-IBE, we show that (n,t)-SK-IBE $\to$ $\mathsf{SKBasicPub}^{hy}$, where $\mathsf{SKBasicPub}^{hy}$ is a public key encryption scheme based on SK-IBE as defined in [CC05, Section 3.2]. Note that $\mathsf{SKBasicPub}^{hy}$ is not a distributed scheme. Therefore, recalling Lemma 2 and 3 from [CC05] to prove $\mathsf{SKBasicPub}^{hy}$ $\to$ $\mathsf{SKBasicPub}$ and $\mathsf{SKBasicPub}$ $\to$ BDHI respectively we complete the proof of Theorem 6.5. Next, we prove (n,t)-SK-IBE $\to$ $\mathsf{SKBasicPub}^{hy}$.

Lemma 6.6. *Let H_1, H_2 be random oracles. Let $\mathcal{A}_1$ be an IND-ID-CCA adversary that has advantage $\epsilon(\kappa)$ in running time $t(\kappa)$ against (n,t)-SK-IBE. Suppose $\mathcal{A}_1$ makes at most q_E, q_D, and q_{H_1} queries. Then there is an IND-CCA adversary $\mathcal{A}_2$ that has advantage at least $\epsilon(\kappa)/q_{H_1}$ against* $\mathsf{SKBasicPub}^{hy}$. *Its running time is at most $t(\kappa) + c(nq_E + q_D + q_{H_1})$ where c is the average time of exponentiation in $\hat{\mathbb{G}}$.*

Proof. We construct an IND-CCA adversary $\mathcal{A}_2$ that uses $\mathcal{A}_1$ to gain advantage against $\mathsf{SKBasicPub}^{hy}$. (For the definition of $\mathsf{SKBasicPub}^{hy}$, refer to [CC05, Section 3.2].) The game between a challenger and $\mathcal{A}_2$ starts with the challenger running algorithm keygen of $\mathsf{SKBasicPub}^{hy}$ to generate a public key $K_{pub} = \langle \mathcal{G}, g, \hat{g}, g^s, h_0, (h_1, \hat{g}^{\frac{1}{h_1+s}}), \ldots, (h_i, \hat{g}^{\frac{1}{h_i+s}}), \ldots, (h_{q_{H_1}}, \hat{g}^{\frac{1}{h_{q_{H_1}}+s}}), H_2, H_3, H_4 \rangle$. Let $\hat{g}^{\frac{1}{h_0+s}}$ be the corresponding private key. The challenger gives K_{pub} to $\mathcal{A}_2$, which is supposed to launch an IND-CCA attack on $\mathsf{SKBasicPub}^{hy}$ using $\mathcal{A}_1$. $\mathcal{A}_2$ simulates the challenger for $\mathcal{A}_1$ as follows.

Setup: As the distributed PKG setup in SK-IBE is same as that of BF-IBE, we reuse much of the Setup simulation of (n,t)-$\mathsf{BFBasicPub}$ in Lemma 6.4. However, we do not require their $\hat{h}^{\mathtt{ID}}$ computation and g^a is replaced by g^s. The master key finally generated is equal to $s' = s\sum_{P_i \in Q_{Good}} \mu_i + \sum_{P_j \in Q_{Bad}} \nu_i$, where $\mathcal{A}_2$ knows $\nu = \sum_{P_j \in Q_{Bad}} \nu_i$ and $\mu = \sum_{P_i \in Q_{Good}} d_i$. To make the pairs $(h_i, \hat{g}^{\frac{1}{h_i+s}})$ compatible with s', $\mathcal{A}_2$ defines $h'_i = \mu h_i - \nu$ and $\hat{g}' = \hat{g}^{\mu}$. To answer H_1 and key extraction queries for $\mathcal{A}_1$, $\mathcal{A}_2$ uses pairs $(h'_i, \hat{g}'^{\frac{1}{h'_i+s'}})$, where $\mathcal{A}_2$ uses h'_i as a hash value and $\hat{g}'^{\frac{1}{h'_i+s'}}$ as the corresponding private key. Further, $\mathcal{A}_1$ is provided $\hat{g}'$ instead of $\hat{g}$ as a public parameter. $\mathcal{A}_2$ also runs random oracles H_1 and H for $\mathcal{A}_1$, where H is a random oracle required in $\mathrm{NIZKPK}_{\mathsf{DLog}}$.

H_1 **queries:** Same as in [CC05, Section 3.2].

Phase 1 - Extraction Queries: Though private keys in the form of $(h'_i, \hat{g}'^{\frac{1}{h'_i+s'}})$ tuples are available, $\mathcal{A}_2$ has to generate those for $\mathcal{A}_1$ in a distributed way as defined in the private-key extraction protocol. This is non-trivial for $\mathcal{A}_2$ as it has to provide shares of $w = (s'+h'_i)z$ to $\mathcal{A}_2$ without knowing its shares of s'. To achieve this, it first chooses $w \in_R \mathbb{Z}_p^*$ and computes $\hat{g}'^{\frac{w}{h'_i+s'}} = \hat{g}'^{z_w}$, where z_w is the randomness which $\mathcal{A}_2$ wants to obtain from $\mathsf{Random}_{\mathsf{Ped}}$. It then completes the actual $\mathsf{Random}_{\mathsf{Ped}}$ and $\mathsf{Mul}_{\mathsf{Ped}}$ protocols normally by playing the part of good parties. It determines z and z' generated by $\mathsf{Random}_{\mathsf{Ped}}$ using its $n-t$ shares and also knows w_i, w'_i for $P_i \in P_{Bad}$. Using w and w_i for $P_i \in P_{Bad}$, it generates w_i and $\hat{g}^{w_i}$ for all parties. To provide the required $\mathrm{NIZKPK}_{\mathsf{DLog}}$ for $\hat{g}^{w_i}$, $\mathcal{A}_2$ randomly generates challenge τ and response (u_1, u_2), computes commitments (t_1, t_2) and includes an entry $\langle (\hat{g}', \hat{h}, F, P, t_1, t_2), \tau \rangle$ in the hash table of H before forwarding $\pi_{\mathsf{DLog}} = (\tau, u_1, u_2)$ to $\mathcal{A}_1$. Similarly, using $\hat{g}'^{z_w} = \hat{g}'^{\frac{w}{h'_i+s'}}$ and $\hat{g}'^{z_{wi}} = \hat{g}'^{z_i}$ for $P_i \in P_{Bad}$, it generates $\hat{g}'^{z_{wi}}$ for each P_i and provides its $\mathrm{NIZKPK}_{\mathsf{DLog}}$, which results in $\mathcal{A}_1$ generating $\hat{g}'^{\frac{1}{h'_i+s'}}$ as its private key.

The rest of the game and the analysis remains exactly the same as [CC05, Section 3.2]. It is interesting to observe that despite the different master keys (s for $\mathsf{SKBasicPub}^{hy}$ and $s' = s\mu + \nu$ for (n,t)-SK-IBE), the ciphertext queries $C = \langle u, v, w \rangle$ remain the same when transferred from $\mathcal{A}_1$ to the challenger during decryption queries and from the challenger to $\mathcal{A}_1$ during the challenge phase. $\square$

6.3.5 Boneh and Boyen's BB$_1$-IBE

BB$_1$-IBE belongs to the commutative-blinding IBE family. Boneh and Boyen [BB04b] proposed the original scheme with a security reduction to the decisional BDH assumption [Jou02] in the standard model against selective-identity attacks. However, with a practical requirement of IND-ID-CCA security, in the IBCS standard [BM07], Boyen and Martin proposed a modified version of BB$_1$, which is IND-ID-CCA secure in the random oracle model under the BDH assumption. In [Boy08], Boyen rightly claims that for practical applications, it would be preferable to rely on the random-oracle assumption rather than using a less efficient IBE scheme with a stronger security assumption or a weaker attack model. Here, we consider the modified BB$_1$-IBE scheme as described in [Boy08] and [BM07].

In the BB$_1$-IBE setup, the PKG generates a master-key triplet $(\alpha, \beta, \gamma) \in \mathbb{Z}_p^3$ and an associated public key tuple $(g^\alpha, g^\gamma, e(g, \hat{g})^{\alpha\beta})$. A client with identity $\mathtt{ID}$ receives the private key tuple $d_{\mathtt{ID}} = (\hat{g}^{\alpha\beta+(\alpha H_1(\mathtt{ID})+\gamma)r}, \hat{g}^r) \in \hat{\mathbb{G}}^2$, where $H_1 : \{0,1\}^* \to \mathbb{Z}_p$.

Distributed PKG Setup. In [Boy08], Boyen does not include the parameters $\hat{g}$ and $\hat{g}^\beta$ from the original BB$_1$ scheme [BB04b] in his public key, as they are not required during key extraction, encryption or decryption (they are not omitted for security reasons). In the distributed setting, we in fact need those parameters to be public for efficiency reasons; a verifiable distributed computation of $e(g, \hat{g})^{\alpha\beta}$ becomes inefficient otherwise. To avoid key escrow of clients' private-key components ($\hat{g}^r$), we also need $\hat{h}$ and $\mathcal{C}^{(\beta)}_{\langle\hat{h}\rangle}$; otherwise, parts of clients' private keys would appear in public commitment vectors. As in SK-IBE in Section 6.3.4, this extra generator $\hat{h} \in \hat{\mathbb{G}}$ is precomputed using the $\mathsf{Random}_{\mathsf{DLog}}$ protocol. Distributed PKG setup of BB$_1$ involves distributed generation of the master-key tuple (α, β, γ). Distributed PKG node P_i achieves this using the following three $\mathsf{Random}_{\mathsf{DLog}}$ protocol invocations:

$$\begin{aligned}
\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right) &= \mathsf{Random}_{\mathsf{DLog}}(n, t, f, g),\\
\left(\mathcal{C}^{(\beta)}_{\langle \hat{g}\rangle}, \beta_i\right) &= \mathsf{Random}_{\mathsf{DLog}}(n, t, f, \hat{g}),\\
\left(\mathcal{C}^{(\gamma)}_{\langle g\rangle}, \gamma_i\right) &= \mathsf{Random}_{\mathsf{DLog}}(n, t, f, g).
\end{aligned}$$

Here, $(\alpha_i, \beta_i, \gamma_i)$ is the tuple of master-key shares for node P_i. We also need $\mathcal{C}^{(\beta)}_{\langle\hat{h}\rangle}$; each node P_i provides this by publishing $\left(\mathcal{C}^{(\beta)}_{\langle\hat{h}\rangle}\right)_i = \hat{h}^{\beta_i}$ and the associated NIZKPK$_{\equiv DLog}(\beta_i, \hat{g}^{\beta_i}, \hat{h}^{\beta_i})$. The tuple $\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, e(g, \hat{g})^{\alpha\beta}, \mathcal{C}^{(\gamma)}_{\langle g\rangle}, \mathcal{C}^{(\beta)}_{\langle\hat{h}\rangle}\right)$ forms the system public key, where $e(g, \hat{g})^{\alpha\beta}$

can computed from the public commitment entries. The vector $\mathcal{C}^{(\beta)}_{\langle\hat{g}\rangle}$, although available publicly, is not required for any further computation.

Private-key Extraction. The most obvious way to compute a BB$_1$ private key seems to be for P_i to compute $\alpha_i\beta_i+(\alpha_i H_1(\mathtt{ID})+\gamma_i)r_i$ and provide $\hat{g}^{\alpha_i\beta_i+(\alpha_i H_1(\mathtt{ID})+\gamma_i)r_i}$, $\hat{g}^{r_i}$ to the client, who now needs $2t{+}1$ valid shares to obtain her private key. However, $\alpha_i\beta_i+(\alpha_i H_1(\mathtt{ID})+\gamma_i)r_i$ here is not a share of a random degree-$2t$ polynomial. The possible availability of $\hat{g}^{r_i}$ to the adversary creates a suspicion about the secrecy of the master-key share with this method.

For private-key extraction in BB$_1$-IBE with a distributed PKG, we instead use the $\mathsf{Mul}_{\mathsf{BP}}$ protocol in which the client is provided with $\hat{g}^{w_i}$, where $w_i = (\alpha\beta + (\alpha H_1(\mathtt{ID})+\gamma)r)_i$ is a share of random degree t polynomial. The protocol works as follows.

1. Once a client with identity ID contacts all n nodes the system, every node P_i authenticates the client's identity and runs $\left(\mathcal{C}_{\langle\hat{h},\hat{g},\rangle}(r,r'), \mathcal{C}^{(r)}_{\langle\hat{h}\rangle}, \text{NIZKPK}_{\mathsf{DLog}}, r_i, r'_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n,t,f,\hat{h},\hat{g})$. $\mathsf{Random}_{\mathsf{Ped}}$ makes sure that r is uniformly random.

2. P_i computes its share w_i of $w = \alpha\beta + (\alpha H_1(\mathtt{ID}) + \gamma)r$ using protocol $\mathsf{Mul}_{\mathsf{BP}}$ in Eq. 6.5.
$$\left(\mathcal{C}^{(w)}_{\langle g^*\rangle}, w_i\right) = \mathsf{Mul}_{\mathsf{BP}}(n,t,f,g^*,desc,\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle},\alpha_i\right),\left(\mathcal{C}^{(\beta)}_{\langle\hat{h}\rangle},\beta_i\right),\left(\mathcal{C}^{(\gamma)}_{\langle g\rangle},\gamma_i\right),\left(\mathcal{C}^{(r)}_{\langle\hat{h}\rangle},r_i\right))$$
where the list $desc = \langle(1,1,2),(H_1(\mathtt{ID}),1,4),(1,3,4)\rangle$ describes the required binary product under the ordering (α,β,γ,r) of secrets. To justify our choices of commitment generators, we present the pairing-based verification in protocol $\mathsf{Mul}_{\mathsf{BP}}$:
$$e(g^{\alpha_i\beta_i+(\alpha_i H_1(\mathtt{ID})+\gamma_i)r_i},\hat{h}) \stackrel{?}{=} e(g^{\alpha_i},\hat{h}^{\beta_i})e((g^{\alpha_i})^{H_1(\mathtt{ID})}g^{\gamma_i},\hat{h}^{r_i}).$$
For type 2 and 3 pairings, $g^* = g$, as there is no efficient isomorphism from $\mathbb{G}$ to $\hat{\mathbb{G}}$. However, for type 1 pairings, we use $g^* = \hat{h} = \phi^{-1}(h)$. Otherwise, the resultant commitments for w (which are public) will contain the private-key part $g^{\alpha\beta+(\alpha H_1(\mathtt{ID})+\gamma)r}$.

3. Once the $\mathsf{Mul}_{\mathsf{BP}}$ protocol has succeeded, Node P_i generates $\hat{g}^{w_i}$ and $\hat{g}^{r_i}$ and sends those to the client over a secure and authenticated channel.

4. The client interpolates the valid shares to generate her private key ($\hat{g}^{\alpha\beta+(\alpha H_1(\mathtt{ID})+\gamma)r}$, $\hat{g}^{r}$). For type 1 and type 2 pairings, the client can use the pairing-based DDH solving to check the validity of the shares. However, for type 3 pairings, without an efficient mapping from $\hat{\mathbb{G}}$ to $\mathbb{G}$, pairing-based DDH solving can only be employed to verify $\hat{g}^{w_i}$. As a verification of $\hat{g}^{r_i}$, node P_i includes a NIZKPK$_{\equiv DLog}(r_i,\hat{h}^{r_i},\hat{g}^{r_i})$ along with $\hat{g}^{w_i}$ and $\hat{g}^{r_i}$.

As in SK-IBE in Section 6.3.4, online execution of the $\mathsf{Random}_{\mathsf{DLog}}$ computation can be eliminated using batch precomputation of distributed random elements $\left(\mathcal{C}^{(r)}_{\langle\hat{h}\rangle}, r_i\right)$.

Encryption and Decryption. Similar to the PKG setup and the key extraction protocols for BB$_1$-IBE in Section 6.3.5, we use the BB$_1$-IBE version defined in [Boy08] and [BM07] for the encryption and decryption protocols here. Boyen [Boy08] claims IND-ID-CCA security of this system against the BDH assumption. This scheme uses $H'_3 = G_t \times \{0,1\}^{\kappa'} \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$ along with H_1 and H_2 from SK-IBE.

Encryption: To encrypt a message M of some fixed bit length κ' for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^{\kappa'}$, computes $k = (e(g,\hat{g})^{\alpha\beta})^{\sigma}$ and $h_{\mathtt{ID}} = \hat{H}(\mathtt{ID})$, and sends the ciphertext $C = (\rho, \rho_0, \rho_1, t) = (M \oplus H_2(k), g^{\sigma}, (g^{\gamma}(g^{\alpha})^{h_{\mathtt{ID}}})^{\sigma}, \sigma + H'_3(k, \rho, \rho_0, \rho_1))$ to the receiver.

Decryption: To decrypt a ciphertext $C = (\rho, \rho_0, \rho_1, t)$ using the private key $d_{\mathtt{ID}} = (\hat{g}^{\alpha\beta + (\alpha H_1(\mathtt{ID}) + \gamma)r}, \hat{g}^r) = (d_0, d_1)$ (say), the receiver successively computes $k = e(\rho_0, d_0)/e(\rho_1, d_1)$ and $\sigma = t - H_3(k, \rho, \rho_0, \rho_1)$. If $k \neq (e(g,\hat{g})^{\alpha\beta})^{\sigma}$ or $\rho_0 \neq g^{\sigma}$, then the receiver rejects C, else it accepts $M = \rho \oplus H_2(k)$ as a valid message.

Proof of Security. We prove IND-ID-CCA security of the BB$_1$-IBE scheme with the (n,t)-distributed PKG ((n,t)-BB$_1$-IBE) based on the BDH assumption. To the best of our knowledge, an IND-ID-CCA security proof for the modified BB$_1$-IBE scheme has not been published yet and a non-distributed version of our proof is the first to provide IND-ID-CCA security for this protocol.

Theorem 6.7. *Let H, H_1, H_2 and H'_3 be random oracles. Let $\mathcal{A}$ be an IND-ID-CCA adversary that has advantage $\epsilon(\kappa)$ in running time $t(\kappa)$ against (n,t)-BB$_1$-IBE making at most q_E, q_D, q_{H_1}, q_{H_2}, $q_{H'_3}$, and q_{H_4} queries. Then, there exists an algorithm $\mathcal{B}$ that solves the BDH problem in $\mathcal{G}$ with advantage roughly equal to $\epsilon(\kappa)/(q_{H_1}q_{H'_3})$ and running time $O(t(\kappa), q_E, q_D, q_H, q_{H_1}, q_{H_2}, q_{H'_3}, q_{H_4})$.*

Proof. Algorithm $\mathcal{B}$ is given a random BDH problem $\langle g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^c\rangle$ in bilinear group $\mathcal{G}$ as input. Let $D = e(g,\hat{g})^{abc} \in \mathbb{G}_t$ be the solution to this problem. Algorithm $\mathcal{B}$ finds D by interacting with $\mathcal{A}$ as follows:

Setup: $\mathcal{B}$ makes a virtual network of n parties and runs the distributed setup of (n,t)-BB$_1$-IBE using the given BDH instance. Let P_{Bad} be the set of t parties corrupted or owned by $\mathcal{A}$. Let P_{Good} be the set of remaining good parties which will be run by $\mathcal{B}$. $\mathcal{B}$ wants to make sure that the challenge g^a is included in both $g^{\alpha} \in \mathcal{C}^{(\alpha)}_{\langle g\rangle}$ and $g^{\gamma} \in \mathcal{C}^{(\gamma)}_{\langle g\rangle}$, and the challenge $\hat{g}^b$ is included in $\hat{g}^{\beta} \in \mathcal{C}^{(\beta)}_{\langle\hat{g}\rangle}$. Similar to the (n,t)-$\mathsf{FullIdent}$ BF-IBE and (n,t)-SK-IBE proofs, the generated master key tuple $(\alpha, \beta, \gamma) = (\mu_1 a + \nu_1, \mu_2 b + \nu_2, \mu_3 a + \nu_3)$.

Let $\mu_3 a + \nu_3 = -\alpha h^*_{\texttt{ID}} + \alpha'$, where $h^*_{\texttt{ID}} = -\mu_3/\mu_1$ is a challenge identity-hash and $\alpha' = \nu_3 - \nu_1\mu_3/\mu_1 = \alpha h^*_{\texttt{ID}} + \gamma$. α' is completely random as the μ and ν values are not under $\mathcal{B}$'s control. Finally, $\mathcal{B}$ outputs $\left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, e(g,\hat{g})^{\alpha\beta}, \mathcal{C}^{(\gamma)}_{\langle g\rangle}, \mathcal{C}^{(\beta)}_{\langle \hat{h}\rangle}, \text{NIZKPK}_{\equiv DLog}\right)$ as the system public key.

H_1 **queries:** Before initializing H_1^{list}, $\mathcal{B}$ chooses $j \in_R \{1, \ldots, q_{H_1}\}$. When $\mathcal{A}$ queries H_1 for $\texttt{ID}_i$, $\mathcal{B}$ proceeds as follows: if $i \neq j$, it picks $h_{\texttt{ID}_i} \in_R \mathbb{Z}_p$, adds a tuple $\langle \texttt{ID}_i, h_{\texttt{ID}_i}\rangle$ and gives back $h_{\texttt{ID}_i}$ to $\mathcal{A}$. If $i = j$, it sets $\langle \texttt{ID}_j, h^*_{\texttt{ID}}\rangle$. Note that multiple queries for the same identity are answered with the corresponding entry in its H_1^{list}. Further, the output of H_1 is uniformly distributed in $\mathbb{Z}_p$ and independent of $\mathcal{A}$'s view.

H_2 **and** H'_3 **queries:** Initially, these lists are empty. When a query for H_2 or H'_3 arrives, $\mathcal{B}$ first checks if an entry for the query input already exists in the corresponding list. If it is present, $\mathcal{B}$ responds with the associated response, else $\mathcal{B}$ sends a random element of the appropriate size as its response, and adds an input and response tuple in the oracle list. The corresponding (random) oracle list entries look as follows: $H_2^{list}(\mathbb{G}_t, \{0,1\}^{\kappa'}) = \langle k_i, h_{k_i}\rangle$, and $H_3^{\prime list}(\mathbb{G}_t, \{0,1\}^{\kappa'}, \mathbb{G}, \mathbb{G}, \mathbb{Z}_p) = \langle k_i, \rho_i, \rho_{0i}, \rho_{1i}, h_{3i}\rangle$, where κ' is the message length.

Phase 1 **- Extraction Queries:** When $\mathcal{A}$ asks for the private key for $\texttt{ID}_i$, $\mathcal{B}$ first gets $H_1(\texttt{ID}_i) = h_{\texttt{ID}_i}$. If $i = j$, then $\mathcal{B}$ aborts the game and the attack fails. If $i \neq j$, $\mathcal{B}$ starts the distributed private-key extraction protocol by running $\left(\mathcal{C}_{\langle \hat{h}, \hat{g},\rangle}(r, r'), r_i, r_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n, t, f, \hat{h}, \hat{g})$. $\mathcal{B}$ knows r, r' as well as shares of the nodes as it runs $n - t$ nodes. It then computes $\hat{h}^{\tilde{r}} = \frac{\hat{h}^r}{(\hat{h}^\beta)^{\Delta h}} = \hat{h}^{r - \beta/\Delta h}$ where $\Delta h = h_{\texttt{ID}_i} - h^*_{\texttt{ID}}$. Using $\hat{h}^{\tilde{r}}$ and the t adversary commitments $\hat{h}^{r_i}$ for $i \in P_{Bad}$, $\mathcal{B}$ computes the commitments $\mathcal{C}^{(\tilde{r})}_{\langle \hat{h}\rangle}$. To provide the required $\text{NIZKPK}_{\mathsf{DLog}}$ for each entry in $\mathcal{C}^{(\tilde{r})}_{\langle \hat{h}\rangle}$, $\mathcal{A}$ randomly generates challenge τ_i and response (u_{1i}, u_{2i}), computes commitments (t_{1i}, t_{2i}) and includes an entry $\langle (\hat{g}', \hat{h}, \hat{h}^{\tilde{r}_i}, \hat{h}^{r_i}\hat{g}^{r'_i}, t_{1i}, t_{2i}), \tau_i\rangle$ in the hash table of H before forwarding $\pi_{\mathsf{DLog}} = (\tau_i, u_{1i}, u_{2i})$ to $\mathcal{A}$.

$\mathcal{B}$ then computes $d'_0 = (g^*)^{-\beta\alpha'/\Delta h}(g^*)^{\alpha r \Delta h + \alpha' r} = (g^*)^{\alpha\beta + (\alpha h_{\texttt{ID}_i} + \gamma)\tilde{r}}$ and using known shares of α_i, β_i and γ_i for $P_i \in P_{Bad}$, it runs $\mathsf{Mul}_{\mathsf{BP}}$ for $w = \alpha\beta + (\alpha h_{\texttt{ID}_i} + \gamma)\tilde{r}$. Note that $\mathcal{B}$ does not know its shares, but it can compute their commitments using d'_0 and the inputs from P_{Bad}. With its (n, t) subshares, it also knows the final shares w_i for $P_i \in P_{Bad}$. It then computes the required private key shares $\hat{g}^{w_i}$ and $\hat{g}^{\tilde{r}_i}$ for $P_i \in P_{Good}$ and forwards them to $\mathcal{A}$.

Phase 1 **- Decryption Queries:** $\mathcal{B}$ answers $\mathcal{A}$'s decryption queries $(\texttt{ID}_i, C_i)$ as follows. $\mathcal{B}$ first gets $H_1(\texttt{ID}_i) = h_{\texttt{ID}_i}$. If $i \neq j$, $\mathcal{B}$ obtains the private key $(\hat{g}^{\alpha\beta + (\alpha h_{\texttt{ID}_i} + \gamma)\tilde{r}}, \hat{g}^r)$ and decrypts $C_i = (t_i, \rho_i, \rho_{0i}, \rho_{1i})$. If $i = j$, then $\mathcal{B}$ cannot compute the private key and it uses H_2 and H'_3 instead. $\mathcal{B}$ searches $H_3^{\prime list}$ for $\langle \cdot, \rho_i, \rho_{0i}, \rho_{1i}\rangle$. If this tuple belongs to a valid ciphertext by $\mathcal{A}$, then there must be one or more corresponding entries in $H_3^{\prime list}$. For each such entry, retrieve k_i and the hash value h'_{3i}. Compute $s_i = t_i - h'_{3\mathtt{ID}_i}$ and check if the component-wise equality $(k_i, \rho_{0i}) \overset{?}{=} (e(g, \hat{g})^s, g^s)$ holds. As $e(g, \hat{g})$, g and ρ_{0i} are fixed for a query, this equality only holds for a single or no k_i value and correspondingly a single or no entry in $H_3'^{list}$. If there is no such entry, then $\mathcal{B}$ discards the ciphertext and aborts, else $\mathcal{B}$ searches for k_i in H_2^{list}. If there is no entry, then $\mathcal{B}$ adds a random entry h_{2i} for k_i in H_2^{list}. Finally, it returns the plaintext M as $M = \rho_i \oplus h_{2i}$.

Challenge: $\mathcal{A}$ outputs an identity $\mathtt{ID}_{ch}$ and two messages M_0 and M_1. If $\mathtt{ID}_{ch} \neq \mathtt{ID}_j$, then it aborts the game and the attack fails, else $\mathcal{B}$ sends $(\rho_b \in_R \{0,1\}^{\kappa'}, \rho_{0b} = g^c, \rho_{1b} = (g^c)^{\alpha'}, t_b \in_R \mathbb{Z}_p)$ as a challenge ciphertext C_b to $\mathcal{A}$.

Phase 2 - Extraction Queries: $\mathcal{B}$ proceeds as in Phase 1, expect the extraction query for $\mathtt{ID}_{ch}$ is rejected.

Phase 2 - Decryption Queries: $\mathcal{B}$ proceeds as in Phase 1, expect the decryption query for $\langle \mathtt{ID}_{ch}, C_b \rangle$ is rejected.

Guess: $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$. Now, there must be one or more entries for $\langle \cdot, \rho_b, \rho_{0b}, \rho_{1b} \rangle$ in $H_3'^{list}$. $\mathcal{B}$ randomly picks one of those tuples $\langle k_i, \rho_i, \rho_{0i}, \rho_{1i} \rangle$ and returns k_i as its answer D.

For a random BDH problem $\langle g, \hat{g}, g^a, \hat{g}^a, \hat{g}^b, g^c \rangle$ in bilinear group $\mathcal{G}$, $\mathcal{A}$'s view is identical to its view in a real attack game. It is easy to observe that B outputs the correct D with probability $\epsilon(\kappa)/(q_{H_1} q_{H_3'})$. □

Using a more expensive DKG protocol with uniformly random output, all of our proofs would become relatively simpler. However, it is important to note that our use of DKG without uniformly random output does not affect the security reduction factor in any proof. This is something not achieved by the known previous protocols with non-uniform DKG such as threshold Schorr signatures [GJKR07]. Further, we do not discuss the liveness and correctness properties for our asynchronous protocols as liveness and correctness of all the distributed primitives provides liveness and correctness for the distributed PKG setup and distributed key extraction protocols. Finally, for simplicity of the discussion, it would have been better to combine the three distributed PKG proofs. However, that looks difficult, if not impossible, as the distributed computation tools used in these distributed PKGs and the original IBE security proofs vary a lot from a scheme to scheme.

6.4 Comparing Distributed PKGs

In this section, we concentrate on the performance of the setup and key extraction procedures of the three distributed PKGs defined in Section 6.3. For a detailed comparison

Table 6.1: Operation count and key sizes for distributed PKG setups and distributed private-key extractions (per key)

	BF-IBE Setup	**BF-IBE** Extraction	**SK-IBE** Setup	**SK-IBE** Extraction	**BB_1-IBE** Setup	**BB_1-IBE** Extraction
Operation Count						
Generator h or $\hat{h}$	X		✓		✓	
HybridDKG-Sh[a]						
(precomputed)	-	0	-	1^P	-	1^P
(online)	1^D	0	1^D	1^P	3^D	1^D
Parings						
@PKG Node	0	0	0	$2n$	1^b	$2n$
@Client	-	$2(2t+2)$	-	0	-	$2n^b$
NIZKPK	0	0	0	$2n$	n^b	$2n^b$
Interpolations	0	1	0	2	1	2
Key Sizes						
PKG Public Key	$(n+2)\mathbb{G}^c$		$(n+3)\mathbb{G}$		$(2n+3)\mathbb{G}$, $(n+2)\hat{\mathbb{G}}$, $(1)\mathbb{G}_T$	
Private-key Shares	$(2t+1)\hat{\mathbb{G}}^c$		$(3n)\mathbb{Z}_p$, $(3n+1)\hat{\mathbb{G}}$		$(2n)\mathbb{Z}_p{}^b$, $(2n)\hat{\mathbb{G}}$	

[a]For HybridDKG-Sh D indicates use of DLog commitments, while P indicates Pedersen commitments.

[b]For type 1 and 2 pairings, n NIZKPKs can be replaced by $2n$ extra pairings and the $2n$ $\mathbb{Z}_p$ elements are omitted from the private-key shares.

[c]For type 2 parings, the groups used for the PKG public key and the private-key shares are interchanged.

of the encryption and decryption algorithms of BF-IBE, SK-IBE and BB_1-IBE, we refer readers to the survey by Boyen [Boy08]. The general recommendations from this survey are to avoid SK-IBE and other exponent-inversion IBEs due to their reliance on the strong BDHI assumption, and that BB_1-IBE and BF-IBE both are good, but BB_1-IBE can be a better choice due to BF-IBE's less efficient encryption.

Table 6.1 provides a detailed operation count and key size comparison of our three distributed PKGs. We count HybridDKG-Sh instances, pairings, NIZKPKs and interpolations, and list public and private key sizes. We leave aside the comparatively small exponentiations and other group operations. As mentioned in Section 6.3.5, for BB_1-IBE, with bilinear pairings of type 1 and 2, there is a choice that can be made between using n NIZKPKs and $2n$ pairing computations. The table shows the NIZKPK choice (the only option for type 3 pairings), and footnote b shows where NIZKPKs can be traded off for pairings. As discussed in Section 6.3.3, for curves with type 2 pairings, an efficient algorithm for hash-to-$\hat{\mathbb{G}}$ is not available and we have to interchange the groups used for the system public-key shares and client private-key shares. Footnote c indicates how that affects the key sizes.

In Table 6.1, we observe that the distributed PKG setup and the distributed private-key extraction protocols for BF-IBE are significantly more efficient than those for SK-IBE and BB_1-IBE. Importantly, for BF-IBE, distributed PKG nodes can extract a key for a client without interacting with each other, which is not possible in the other two schemes; both BB_1-IBE and SK-IBE require at least one DKG instance for every private-key extraction; the second required instance can be batch precomputed. Therefore, for IBE applications in the random oracle model, we recommend the use of the BF-IBE scheme, except in situations where private-key extractions are rare and efficiency of the encryption step is critical to the system. For such applications, we recommend BB_1-IBE as the small efficiency gains in the distributed PKG setup and extraction protocols of SK-IBE do not well compensate for the strong security assumption required. BB_1-IBE is also more suitable for type 2 and 3 pairings, where an efficient map-to-group hash function $\hat{H}$ is not available. Further, BB_1-IBE can also be proved secure in the standard model with selective-identity attacks. For applications demanding security in the standard model, our distributed PKG for BB_1-IBE also provides a solution to the key escrow and single point of failure problems, using pairings of type 1 or 2.

Proactive Security and Group Modification. We observe that the proactive security and group modification protocols defined in Chapter 5, for the DKG protocol used in our distributed PKGs, are directly applicable to our distributed PKGs. We suggest the use of these protocols to achieve proactive security of our master keys and group modification of our PKGs. Note that this is possible only due to the nature of the master keys for the three IBE schemes that we use. All master key elements in these three schemes belong to $\mathbb{Z}_p$, which is also the output domain for the DKG protocol. In contrast to the three IBEs that we consider, we leave as an open problem the possibility of providing proactive security and group modification protocols to the master keys for IBE schemes such as the original BB_1-IBE [BB04b] or Waters' IBE [Wat05].

In this chapter, we designed and compared distributed PKG setup and private key extraction protocols for BF-IBE, SK-IBE, and BB_1-IBE. We observed that the distributed PKG implementation for BF-IBE is the most simple and efficient among all and we suggest its use when the system can support its relatively costly encryption step. In the next chapter, we use it to define an onion routing circuit construction in the identity-based setting. It solves the key escrow problem in this privacy-enhancing setting over the Internet.

Chapter 7

Pairing-Based Onion Routing and Extensions

7.1 Preliminaries

Chaum [Cha81] introduced the concept of *digital pseudonyms* and *mix networks* in 1981. Over the years, a significant number of anonymity networks have been proposed and implemented. Common to many of them is *onion routing* [RSG98], a technique whereby a message is wrapped in multiple layers of encryption, forming an *onion.* A common realization of an onion routing system is to arrange a collection of *onion routers* (abbreviated ORs, also called *hops* or *nodes*) that will relay traffic for users of the system. Users then randomly choose a path through the network of onion routers and construct a *circuit*—a sequence of nodes which will route traffic. After the circuit is constructed, each of the nodes in the circuit shares a symmetric key with the user, which will be used to encrypt the layers of future onions. Upon receiving an onion, each node decrypts one of the layers, and forwards the message to the next node.

Pairing-based cryptography has drawn an overwhelming amount of research attention in the last decade. In one of the pioneering works in the field, Sakai, Ohgishi, and Kasahara [SOK00] presented a non-interactive key agreement scheme in the identity-based setting. As the first contribution of this chapter, we enhance their protocol to develop provably secure one- or two-way privacy-preserving authentication and key agreement schemes [KZG07]. After one-way authentication between Alice (who will remain anonymous) and Bob (who is to be authenticated), Alice has confirmed Bob's identity and Bob learns nothing about Alice, except perhaps that she is a valid user of a particular system. In a two-way scheme, each user can confirm the other is a valid user without learning who the other is.

We then use our one-way anonymous key agreement protocol to build onion routing circuits for anonymity networks like Tor [DMS04] and prove security-related properties of the new construction called *pairing-based onion routing* (PB-OR). Our PB-OR protocol, which first appeared in [KZG07], constructs a circuit in a single pass and also provides a practical way to achieve *eventual forward secrecy* (forward secrecy once the private keys are rotated, as defined in [ØS07]). Observing the performance trade-off between eventual and *immediate forward secrecy* (traditional forward secrecy, effective once any session ends) in onion routing circuit construction, we develop a λ-pass circuit construction [KZG09], which obtains immediate forward secrecy at λ nodes by incorporating λ single-pass circuit constructions. Finally, we define a practical generic onion routing circuit construction protocol that achieves security in the universal composability (UC) framework [Can01] and apply that protocol to the PB-OR protocol to define its UC-secure version. [KG10b] We achieve this using *Sphinx*, an efficient message format for mix networks, defined by Danezis and Goldberg [DG09].

The performance of PB-OR, λ-pass PB-OR and their UC-secure versions surpass that of Tor, requiring significantly less computation, fewer network communications and shorter message sizes. Further, they do not require the public keys of onion routers to be authenticated and consequently, reduce the load on directory servers which improves the scalability of anonymity networks.

However, as we discussed in the previous chapter, key escrow and single point of failure problems are inherent to IBC. A private-key generator (PKG) in IBC can decrypt all messages encrypted for its clients. As our schemes use the same setup as BF-IBE, to mitigate this problem, we use our distributed PKG for BF-IBE that we developed in Section 6.3.3, where the master key is generated in a completely distributed way.

In the rest of this section, we cover the previous work related to anonymity networks and pairing-based key exchange. We describe and prove the anonymous key agreement protocol in the BF-IBE setting in Section 7.2, and build an onion routing circuit construction (PB-OR) protocol based on it in Section 7.3. In Section 7.4, we describe our λ-pass PB-OR circuit construction. In Section 7.5, we reduce the sizes of PB-OR circuit construction messages using the Sphinx methodology. In Section 7.6, we compare the computational and communication costs and the message sizes of our constructions to those of Tor.

Related Work. The concept of onion routing plays a key role in many efforts to provide anonymous communication [Dai98, DMS04, FM02, RSG98, RP02] while a number of other papers discuss formalizations and the security of onion routing [CL05, MVdV04, Möl03, STRL00]. To date, the largest onion routing system is Tor, which has thousands of onion routers and hundreds of thousands of users [Tor10]. These numbers (and their growth) underscore the demand for anonymity online.

In the original Onion Routing project [GRS96, RSG98, STRL00] (which was superseded by Tor) circuit construction was done as follows. The user created an onion where each layer contained the symmetric key for one node and the location of the next node, all encrypted with the original node's public key. Each node decrypts a layer, keeps the symmetric key and forwards the rest of the onion along to the next node. The main drawback of this approach is that it does not provide forward secrecy (as defined in [DMS04]). Suppose a circuit is constructed from the user to the sequence of nodes $A \Leftrightarrow B \Leftrightarrow C$, and that A is malicious. If A records the traffic, and at an arbitrary time in the future compromises B (at which point he learns the next hop is C), and then compromises C, the complete route is known, and A learns who the user has communicated with. A possible fix for this problem is to frequently change the public keys of each node. This limits the amount of time A has to compromise B and C, but requires that the users of the system frequently contact the directory server to retrieve authentic keys.

Later systems constructed circuits incrementally and interactively (this process is sometimes called *telescoping*). The idea is to use the node's public key only to initiate a communication during which a temporary session key is established via the DH key exchange. Tor constructs circuits in this way, using the Tor authentication protocol (TAP). TAP is described and proven secure in [Gol06].

Trade-offs exist between the two methods of constructing circuits. Forward secrecy is the main advantage of telescoping, but telescoping also handles nodes that are not accepting connections; if the third node is down during the construction of a circuit, for example, the first two remain, and the user only needs to choose an alternate third. Information about the status and availability of nodes is therefore less important. The drawback of telescoping is cost; establishing a circuit of length ℓ requires $O(\ell^2)$ network communications, and $O(\ell^2)$ symmetric encryptions/decryptions. In the existing Tor implementation, this results in a latency of a few seconds. As a result, users find Tor to be very slow for their usual communication over the Internet, and employ it only in situations where their anonymity is indispensable to them. Efficiency is essential for widespread use of anonymity networks; therefore, defining a more efficient onion routing circuit construction protocol is an interesting research problem.

Øverlier and Syverson [ØS07] improve the efficiency of telescoping-based circuit construction using a half-certified DH key exchange [MOV97, Sec. 12.6]. They further define an efficient single-pass circuit construction and a few variants using the pre-distributed DH values; we call this scheme *Tor-preDH*. The proposed variants offer different levels of forward secrecy, which are traded off against computation and communication. For example, their eventual forward secret variants use frequent rotation of nodes' public keys, presenting the same issues as in first-generation onion routing; their immediate forward secrecy variant uses the same amount of communication as the current Tor ($O(\ell^2)$), but less computation.

In related efforts, Camenisch and Lysyanskaya [CL05] formally define the requirements of a secure onion routing construction in the UC framework and present a generic construction of onion routing circuits. Although formally secure, their construction is less efficient than other constructions due to the additional mechanisms required to prove security in the UC framework. Our Sphinx-based design achieves UC security without compromising the efficiency of the circuit construction.

The work of Okamoto and Okamoto [OO05] presents schemes for anonymous authentication and key agreement. In Rahman *et al.* [RIO$^+$06], an anonymous authentication protocol is presented as part of an anonymous communication system for mobile *ad-hoc* networks. The protocols in both papers are complex, and limited motivation is given for design choices. Further, both papers neglect to discuss the security of their proposed protocols. Our anonymous key agreement protocols are a great deal simpler than previous protocols. This allows them to be more easily understood, and simplifies the discussion of their security. Huang [Hua07] presents a pseudonym-based encryption scheme similar to our anonymous key agreement protocol in Section 7.2, but differs in its method of private-key extraction as well as in the motivation behind its use. All these protocols owe a lot to the identity-based non-interactive key exchange protocol of Sakai et al. [SOK00] (SOK key agreement) and we begin the next section by reviewing it.

7.2 Anonymous Key Agreement in the BF-IBE Setting

In one of the pioneering works of pairing-based cryptography, Sakai *et al.* suggested an identity-based non-interactive key agreement scheme using bilinear pairings [SOK00]. In this section, we extend this key agreement scheme. We replace the identities of the participants by pseudonyms; the resulting scheme provides unconditional anonymity to participating users.

Consider the usual bilinear pairing setting having groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ of order p such that $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$. In the BF-IBE setup, as we discuss in Section 6.3.3, a PKG generates private keys (d_i) for clients using the clients' well-known identities ($\mathtt{ID}_i$) and a master secret $s \in \mathbb{Z}_p^*$. A client with identity $\mathtt{ID}_i$ receives the private key $d_i = \hat{H}(\mathtt{ID}_i)^s \in \hat{\mathbb{G}}^*$, where $\hat{H} : \{0,1\}^* \rightarrow \hat{\mathbb{G}}^*$ is a full-domain cryptographic hash function.

Sakai *et al.* observed that, if the pairing is symmetric (type 1), any two clients of the same PKG in a BF-IBE setup can compute a shared key using only the identity of the other participant and their own private keys. Only the two clients and the PKG can compute this key. For two clients with identities $\mathtt{ID}_A$ and $\mathtt{ID}_B$, the shared key is given by $K_{A,B} = e(h_A, h_B)^s = e(h_A, d_B) = e(d_A, h_B)$ where $h_A = \hat{H}(\mathtt{ID}_A)$ and $h_B = \hat{H}(\mathtt{ID}_B)$.

Note that this protocol does not work with pairings of type 2 as it requires the feasibility of hashing to both groups $\mathbb{G}$ and $\hat{\mathbb{G}}$. For type 3 pairings, the protocol does not work in its current form as the private keys of the two participants must be in different groups for the protocol to work. In the absence of an isomorphism between $\mathbb{G}$ and $\hat{\mathbb{G}}$ with type 3 pairings, the PKG will need to generate two private keys for each user for the protocol to work.

7.2.1 Constructions

For pairings of type 1, we observe that by replacing the identity hashes with pseudonyms generated by users, a key agreement protocol with unconditional anonymity is possible. In this protocol, a participant cannot determine the identity of the other participant. Each client can, on her own, randomly generate many possible pseudonyms and the corresponding private keys.

Suppose Alice, with (identity, private key) pair $(\mathtt{ID}_A, d_A)$, is seeking anonymity. She generates a random $r_A \in \mathbb{Z}_p^*$ and creates the pseudonym and corresponding private key $(\alpha_A = h_A^{r_A} = \hat{H}(\mathtt{ID}_A)^{r_A}, d_A^{r_A} = \alpha_A^s)$. In a key agreement protocol, she sends the pseudonym α_A instead of her actual identity to another participating client, who may or may not be anonymous. For two participants (say Alice and Bob) with pseudonyms α_A and α_B, the shared session key is given as

$$K_{A,B} = e(\alpha_A, \alpha_B)^s = e(h_A, h_B)^{r_A r_B s}$$

where r_A and r_B are random numbers generated respectively by Alice and Bob. If Bob does not wish to be anonymous, he can just use $r_B = 1$ instead of a random value, resulting in $\alpha_B = h_B$. If persistent pseudonymity is desired instead of anonymity, the random values can easily be reused.

Two participants can perform a session key agreement by exchanging pseudonyms. Further, two participants can also perform an authenticated key agreement by modifying any secure symmetric-key based mutual authentication protocol and simply replacing their identities by their pseudonyms. Finally, this two-way anonymous key agreement protocol does not work with the pairings of type 2 and 3 for the reasons same as those for the SOK key agreement scheme.

One-Way Anonymous Key Agreement. Anonymous communication often requires anonymity for just one of the participants; the other participant works as a non-anonymous service provider and the anonymous participant needs to confirm the service provider's identity. In this one-way anonymous key agreement protocol, the service provider uses her actual identity rather than a pseudonym. Further, in this one-way anonymity setting two participants can agree on a session key in a non-interactive manner.

Suppose Bob is a client of a PKG and functions as a service provider for Alice and other users. Here, to make protocol suitable for all three types of pairings, we assume that the PKG provides a public pair (g, g^s) for the users. Alice, who wishes to remain anonymous to Bob, uses this public pair instead of her identity $\mathtt{ID}_A$ and private key d_A.

1. Alice computes $h_B = \hat{H}(\mathtt{ID}_B)$ for Bob (the service provider). She chooses a random integer $r_A \in \mathbb{Z}_p^*$, generates the corresponding pseudonym $\alpha_A = g^{r_A}$ and private key $(g^s)^{r_A} = \alpha_A^s$, and computes the session key $K_{A,B} = e(\alpha_A^s, h_B) = e(g, h_B)^{sr_A}$. She sends her pseudonym α_A to Bob.
2. Bob, using α_A and his private key d_B, computes the session key $K_{A,B} = e(\alpha_A, d_B) = e(g, h_B)^{sr_A}$.

Note that in step 1, Alice can also include a message for Bob symmetrically encrypted with the session key; we will use this in Section 7.3. Note also that in practice, the session key is often derived from $K_{A,B}$, and is not just $K_{A,B}$ itself. We only need a hash function mapping to one of $\mathbb{G}$ or $\hat{\mathbb{G}}$ here. Therefore, the protocol works for all three types of pairings. For type 2 pairings, in the absence of a hash function mapping to $\hat{\mathbb{G}}$, the groups for the public key and a service provider's private key have to be interchanged.

Key Authentication and Confirmation. In most one-way anonymous communication situations, it is also required to authenticate the non-anonymous service provider. With the non-interactive protocols of this section, the key is implicitly authenticated; Alice is assured that only Bob can compute the key. If Alice must be sure Bob has in fact computed the key, explicit key confirmation can be achieved by incorporating any symmetric-key based challenge-response protocol.

7.2.2 Analysis

We make the following claims in the random oracle model.

Unconditional Anonymity. It is impossible for the other participant in a protocol run, the PKG, or any third party to learn the identity of an anonymous participant in a protocol run.

Session Key Secrecy. It is infeasible for anyone other than the two participants or the PKG to determine a session key generated during a protocol run.

No Impersonation. It is infeasible for a malicious client of the PKG to impersonate another non-anonymous client in a protocol run. In the case of persistent pseudonymity, it is not feasible for a malicious entity to communicate using a different entity's pseudonym.

Next, we prove each of our claims.

Unconditional Anonymity. Here, we prove that it is impossible for an adversary $\mathcal{A}$ to learn the identity of an anonymous participant in a protocol run. For ease of comprehension, we first briefly outline the argument in an informal manner. In the one-way anonymous key agreement setting, it is easy as the anonymous clients do not use their identities. In the two-way anonymous key agreement protocol, for an anonymous client with identity $\mathtt{ID}_A$, the pseudonym $\alpha_A = h_A^{r_A} \in \mathbb{G}$ is the only parameter exchanged during the protocol that is derived from her identity. Because $\hat{\mathbb{G}}$ is a cyclic group of prime order, h_A is a generator, so exponentiating with the random r_A blinds the underlying identity from the adversary $\mathcal{A}$, which can be the other participant in the protocol run, the PKG for the system or any third party. To formalize our proof, we consider the following game between an adversary and a challenger.

Setup. The adversary $\mathcal{A}$ publishes the system parameters: a group $\hat{\mathbb{G}}$ of prime order p and a hash function $\hat{H} : \{0,1\}^* \rightarrow \hat{\mathbb{G}}^*$.

Challenge. $\mathcal{A}$ chooses two identity strings $\mathtt{ID}_A$ and $\mathtt{ID}_B$ and sends them to the challenger. The challenger computes $h_A = \hat{H}(\mathtt{ID}_A)$ and $h_B = \hat{H}(\mathtt{ID}_B)$. He then uniformly at random chooses $r \in \mathbb{Z}_p^*$ and $b \in \{0,1\}$, then

1. if $b = 0$, computes a pseudonym $\alpha = h_A^r$ or
2. if $b = 1$, computes a pseudonym $\alpha = h_B^r$

and sends α to $\mathcal{A}$.

Guess. $\mathcal{A}$ wins the game if she can guess the correct value of b with probability $1/2 + \epsilon(\kappa)$ for a non-negligible function ϵ.

As $\hat{\mathbb{G}}$ is a cyclic prime order group, both h_A and h_B are generators of $\hat{\mathbb{G}}$. For the uniform random element $r \in \mathbb{Z}_p^*$, the pseudonym α equal to h_A^r or h_B^r is also a uniform random element of $\hat{\mathbb{G}}^*$. Therefore, an attacker cannot determine which of the two ways the challenger generated α and consequently cannot guess the value of b with probability greater than 1/2 to win this game. The inability of the attacker to win this game for system parameters of their choosing, even with unbounded computation power, proves our unconditional anonymity claim.

Session Key Secrecy. Dupont and Enge [DE06] prove the security of the SOK key agreement scheme in the random oracle model under the BDH assumption using an analysis technique by Coron [Cor00]. According to their proof, an attacker cannot compute the shared key if the BDH assumption holds and $\hat{H}$ is modelled by a random oracle. Here, we modify their proof to prove that it is infeasible for anyone other than the two participants or the PKG to determine a session key generated during a protocol run of the one-way or two-way anonymous key agreement.

Consider the following game to prove key secrecy in the one-way anonymous case.

Setup. The challenger generates groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ of order p, a cryptographic hash function $\hat{H} : \{0,1\}^* \to \hat{\mathbb{G}}^*$, a bilinear pairing $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ and a master secret $s \in \mathbb{Z}_p^*$.

Extraction Queries. The adversary $\mathcal{A}_1$ issues q extraction queries for identities $\texttt{ID}_1, \texttt{ID}_2, \ldots, \texttt{ID}_q$ to the challenger. The challenger queries $\hat{H}$ to compute the corresponding private keys $\hat{H}(\texttt{ID}_1)^s, \hat{H}(\texttt{ID}_2)^s, \ldots, \hat{H}(\texttt{ID}_q)^s$ and sends them back to $\mathcal{A}_1$.

Challenge. Once $\mathcal{A}_1$ informs the challenger that it has collected enough information, the challenger picks an element $\alpha_A \in \mathbb{G}^*$ and sends it to $\mathcal{A}_1$.

Guess. $\mathcal{A}_1$ outputs a binary string (an identity) $\texttt{ID}_B$ different from the identities $\texttt{ID}_i$ asked in the above extraction queries and $K_{A,B} \in \mathbb{G}_T$.

The attacker's advantage can be defined as

$$\text{Adv}(\mathcal{A}_1) = \Pr[e(\alpha_A, \hat{H}(\texttt{ID}_B))^s = K_{A,B}]$$

We say $\mathcal{A}_1$ (t_1, ϵ_1)-wins the game, if it runs in time at most t_1 and has advantage ϵ_1.

Next, consider the following game to prove key secrecy in the two-way anonymous case and type 1 pairing.

Setup. The challenger generates groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ of order p, a cryptographic hash function $\hat{H} : \{0,1\}^* \to \hat{\mathbb{G}}^*$, a bilinear pairing $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ and a master secret $s \in \mathbb{Z}_p^*$.

Extraction Queries. The adversary $\mathcal{A}_2$ issues q extraction queries for identities $\texttt{ID}_1, \texttt{ID}_2, \ldots, \texttt{ID}_q \in \mathbb{G}$ to the challenger. The challenger queries $\hat{H}$ to compute the corresponding private keys $\hat{H}(\texttt{ID}_1)^s, \hat{H}(\texttt{ID}_2)^s, \ldots, \hat{H}(\texttt{ID}_q)^s$ and sends them back to $\mathcal{A}_2$.

Challenge. Once $\mathcal{A}_2$ informs the challenger that it has collected enough information, the challenger picks two elements $\alpha_A \in \mathbb{G}^*$ and $\alpha_B \in \hat{\mathbb{G}}^*$ and sends them to $\mathcal{A}_2$.

Guess. $\mathcal{A}_2$ outputs $K_{A,B} \in \mathbb{G}_T$.

The attacker's advantage can be defined as

$$\text{Adv}(\mathcal{A}_2) = \Pr[e(\alpha_A, \alpha_B)^s = K_{A,B}]$$

We say $\mathcal{A}_2$ (t_2, ϵ_2)-wins the game, if it runs in time at most t_2 and has advantage ϵ_2.

Suppose that there is an adversary $\mathcal{A}_1$ who (t_1, ϵ_1)-wins the one-way anonymous game and an adversary $\mathcal{A}_2$ who (t_2, ϵ_2)-wins the two-way anonymous game. We now show that an algorithm $\mathcal{B}$ can make use of $\mathcal{A}_1$ or $\mathcal{A}_2$ to solve a random instance of the BDH problem.

Theorem 7.1. *Let the hash function H be modelled by a random oracle. Suppose there exist adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathcal{A}_1$ (t_1, ϵ_1)-wins the one-way anonymous protocol security game and $\mathcal{A}_2$ (t_2, ϵ_2)-wins the two-way anonymous protocol security game. Then there exists an algorithm $\mathcal{B}$ which solves the BDH problem*

- *using $\mathcal{A}_1$ with probability $\frac{\epsilon_1}{e(1+q)}$ in time $t_1 + wq + t_T + t_{inv}$ or*
- *using $\mathcal{A}_2$ with probability ϵ_2 in time $t_2 + wq$.*

Here e is the base of natural logarithms, w is a small constant, q is an upper bound on the number of extraction queries performed by an adversary, t_T is the time required for exponentiation in $\mathbb{G}_T$ and t_{inv} is the time required to invert an element of $\mathbb{Z}_p^*$.

Proof. Let $\langle g, \hat{g}, g^a, \hat{g}^a, g^b, \hat{g}^c \rangle$ be a random and uniformly distributed instance of the BDH problem, which algorithm $\mathcal{B}$ receives as input. To find the solution $e(g, \hat{g})^{abc}$, $\mathcal{B}$ simulates the challenger for $\mathcal{A}_1$ or $\mathcal{A}_2$. This means that $\mathcal{B}$ must simulate the random oracle $\hat{H}$ and answer the private key extraction queries by $\mathcal{A}_1$ or $\mathcal{A}_2$. As the steps for $\hat{H}$-queries and extraction queries are the same in both $\mathcal{A}_1$ or $\mathcal{A}_2$, we denote both of them by $\mathcal{A}$.

$\hat{H}$-queries. At any time, $\mathcal{A}$ can query the random oracle $\hat{H}$. To respond to these queries, $\mathcal{B}$ maintains an initially empty list L of quadruples $(X, h, v, \beta) \in \{0,1\}^* \times \hat{\mathbb{G}}^* \times \mathbb{Z}_p^* \times \{0,1\}$. When $\mathcal{A}$ queries for the hash value of some bit-string X_i, algorithm $\mathcal{B}$ responds as follows:

1. If L contains a quadruple (X_i, h_i, v_i, β_i), $\mathcal{B}$ responds by sending h_i.
2. Otherwise, $\mathcal{B}$ generates at random $\beta_i \in \{0,1\}$, so that $\Pr[\beta_i = 0] = \delta$, where δ depends on B's choice for the attacker ($\mathcal{A}_1$ or $\mathcal{A}_2$) and will be determined below.
3. Algorithm $\mathcal{B}$ picks a random $v_i \in \mathbb{Z}_p^*$. If $\beta_i = 0$, set $h_i = \hat{g}^{v_i}$, else set $h_i = (\hat{g}^c)^{v_i}$. Note that either way, h_i is uniformly random in $\hat{\mathbb{G}}^*$ and independent of $\mathcal{A}$'s current view.

4. Finally, algorithm $\mathcal{B}$ adds the quadruple (X_i, h_i, v_i, β_i) to the list L and responds with h_i.

Extraction queries. $\mathcal{A}$ can ask for extraction queries for identity strings. For an input string $\mathtt{ID}_i$ for private key extraction, $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ runs the above $\hat{H}$-query algorithm for input $X_i = \mathtt{ID}_i$ to obtain $(\mathtt{ID}_i, h_i, v_i, \beta_i)$.
2. If $\beta_i = 1$ then $\mathcal{B}$ reports failure.
3. Otherwise, $\mathcal{B}$ computes the private key $(\hat{g}^a)^{v_i} = h_i^a$ and sends it to algorithm $\mathcal{A}$.

Challenge. After completing the extraction queries, B challenges $\mathcal{A}_1$ with $\alpha_A = g^b$ or $\mathcal{A}_2$ with $\alpha_A = g^b$ and $\alpha_B = \hat{g}^c$.

Guess. $\mathcal{A}_1$ outputs $(\mathtt{ID}_B, K_{A,B}) \in \{0,1\}^* \times \mathbb{G}_T$ or A_2 outputs $K_{A,B} \in \mathbb{G}_T$. In the case of adversary $\mathcal{A}_2$, $\mathcal{B}$ outputs $\sigma = K_{A,B}$ as its guess for the solution to the BDH problem. For the adversary $\mathcal{A}_1$, algorithm $\mathcal{B}$ performs following steps:

1. $\mathcal{B}$ obtains the quadruple $(\mathtt{ID}_B, h_B, v_B, \beta_B)$ from the list L. Absence of the quadruple $(\mathtt{ID}_B, h_B, v_B, \beta_B)$ in the list L indicates that $\mathcal{A}_1$ did not ask the random oracle for $\hat{H}(\mathtt{ID}_B)$. As the probability of the adversary's success in this case is negligible[1], we safely assume the presence of the quadruple $(\mathtt{ID}_B, h_B, v_B, c_B)$.
2. If $\beta_B = 1$, $\mathcal{B}$ outputs $\sigma = K_{A,B}^{v_B^{-1}}$ as its guess for the BDH instance.
3. If $\beta_B = 0$, $\mathcal{B}$ reports failure.

Suppose that $\mathcal{B}$ does not report failure and outputs σ while using $\mathcal{A}_1$. As $\beta_B = 1$, $\hat{H}(\mathtt{ID}_B) = (\hat{g}^c)^{v_B}$ and with probability ϵ_1, $\sigma = e(g^b, (\hat{g}^c)^{v_B})^{av_B^{-1}} = e(g, \hat{g})^{abc}$. Therefore $\mathcal{B}$ will guess correctly with probability ϵ_1, when it does not abort. The probability that $\mathcal{B}$ does not abort while extracting a single private key query is δ; for q queries, the probability is δ^q. The probability that $\mathcal{B}$ does not abort while guessing the BDH solution is $1 - \delta$. Therefore, the overall probability of non-abortion is $\delta^q(1-\delta)$. Maximizing this probability, the optimal value can be obtained at $\delta = \frac{q}{1+q}$ and by choosing the value of δ optimally, the overall probability of non-abortion is $\frac{q^q}{(1+q)^{q+1}}$. Therefore, $\mathcal{B}$ outputs the correct solution to the BDH instance with probability at least $\frac{\epsilon_1 q^q}{(1+q)^{q+1}} \geq \frac{1}{e(1+q)}$ as $(1 - \frac{1}{q+1})^q \geq 1/e$. The solution is computed in time $t_1 + wq + t_T + t_{inv}$, where t_1 is the time required by $\mathcal{A}$, w is the time required to answer an extraction query (generate a random element r and compute

[1] If $\mathcal{A}_1$ does not query the random oracle for $\hat{H}(\mathtt{ID}_B)$, the probability it can guess this value is negligible. As there is a bijection between x and $e(\alpha_A, x)$ for a given α_A, the probability that $\mathcal{A}_1$ can output $K_{A,B} = e(\alpha_A, \hat{H}(\mathtt{ID}_B))$ is also negligible. Thus, $\mathcal{A}_1$'s advantage in this case is negligible.

$(\hat{g}^a)^r)$, q is an upper bound on the number of such queries, and $t_T + t_{inv}$ is the time to compute an exponentiation of $K_{A,B} \in \mathbb{G}_T$ and to invert an element of $\mathbb{Z}_p^*$ in the guessing phase.

For adversary $\mathcal{A}_2$, we simply set the value of δ to 1. Suppose that $\mathcal{B}$ does not report failure and outputs σ while using $\mathcal{A}_2$. With probability ϵ_2, $\sigma = e(g^b, \hat{g}^c)^a = e(g, \hat{g})^{abc}$, which is the correct solution to the BDH problem. The solution is computed in time $t_2 + wq$. □

Note that it is possible to prove the security of the two-way anonymous key agreement protocol without random oracles, if we do not consider the query extraction phase. Assume that only one identity hash and private key pair (g, g^s) is publicly available and each user uses the same pair to generate a pseudonym and corresponding private key. Given an adversary $\mathcal{A}$ to (t, ϵ)-compute $K_{A,B} = e(\alpha_A, \alpha_B)^s$ when challenged by α_A and α_B, a random instance $(g, \hat{g}, g^a, \hat{g}^a, g^b, \hat{g}^c)$ of the BDH problem can be solved in time t with probability ϵ by publishing (g, g^a) as the publicly available identity hash and private key and challenging $\mathcal{A}$ with $\alpha_A = g^b$ and $\alpha_B = \hat{g}^c$.

No Impersonation. We claim that it is infeasible for a malicious client of the PKG to impersonate another (non-anonymous) client in a protocol run. To successfully impersonate a non-anonymous participant $\texttt{ID}_N$ in our one-way anonymous key agreement protocol, given a pseudonym and $\texttt{ID}_N$, an adversary needs to determine the corresponding session key. We observe that the adversary game for non-anonymous participant impersonation is the same as the key secrecy game of the one-way anonymous key agreement and consequently the corresponding theorem and proof carry over.

In the case of persistent pseudonymity, we claim that it is not feasible for a malicious entity to communicate using a different entity's pseudonym. Here, the malicious entity needs to find the shared secret key for a persistent pseudonym generated and used by some other anonymous entity and an arbitrary identity or pseudonym for which it does not know the private key. In the one-way anonymous communication protocol, the corresponding adversary game remains the same as that for impersonation of the non-anonymous entity, and in the two-way anonymous case, the game is the same as the one used to prove key secrecy. Consequently, the theorem and proof for the corresponding game are same as those used to prove key secrecy.

Applications of Our Anonymity Schemes. Our anonymous key agreement schemes can be used to perform anonymous communication in any setting having a BF-IBE setup. In recent years, numerous BF-IBE based solutions have been suggested for various practical situations such as *ad-hoc* networks [CL06, KKA03] and delay-tolerant networks [SK05]. Our anonymous key agreement schemes can be used in all of these setups without any extra effort. As an example, we refer readers to our secure anonymous communication scheme for delay-tolerant networks [KZH07].

In this thesis, we focus on a new pairing-based onion routing protocol which achieves forward secrecy and constructs circuits without telescoping. We describe this protocol in the next section.

7.3 Pairing-Based Onion Routing (PB-OR)

Low-latency onion routing requires one-way anonymous key agreement and forward secrecy. In this section, we describe a new pairing-based onion routing protocol using the non-interactive key agreement scheme defined in Section 7.2.1.

Our onion routing protocol has a significant advantage over the original onion routing protocol [GRS96] as well as the protocol used in Tor [DMS04]; it provides a practical way to achieve forward secrecy without building circuits by telescoping. Though this is possible with the original onion routing protocol, that method involves regularly communicating authenticated copies of onion routers' (ORs') public keys to the system users; forward secrecy is achieved by periodically rotating these keys. This does not scale well; every time the public keys are changed *all* users must contact a directory server to retrieve the new authenticated keys. However, our onion routing protocol uses ORs' identities, which users can obtain or derive without repeatedly contacting a central server, thus providing practical forward secrecy without telescoping.

Design Goals and Threat Model. As our protocol only differs from existing onion routing protocols in the circuit construction phase, our threat model is that of Tor. For example, adversaries have complete control over some part (but not all) of the network, as well as control over some of the nodes themselves.

We aim at frustrating attackers from linking multiple communications to or from a single user. Like Tor, we do not try to develop a system secure against a global observer, which can in theory follow end-to-end traffic. Further, it should not be feasible for any node to determine the identity of any node in a circuit other than its two adjacent nodes. Finally, we require eventual forward secrecy: after some amount of time, the session keys used to protect node identities and the contents of messages are irrecoverable, even if all participants in the network are subsequently compromised.

PB-OR Circuit Construction Protocol. An onion routing protocol involves a service provider, a set of onion routers, and users. In our protocol, a user does not build the circuit incrementally via telescoping, but rather in a single pass. The user chooses ℓ ORs from the available pool and generates separate pseudonyms for communicating with each of them. The user computes the corresponding session keys and uses them to construct a message with ℓ nested layers of encryption. This process uses the protocol given in Section 7.2.1 ℓ times.

There are two time-scale parameters in our protocol: the *master key validity period* ($\mathtt{VP}_{\mathtt{MK}}$) and the *private key validity period* ($\mathtt{VP}_{\mathtt{PK}}$). Both of these values relate to the forward secrecy of the system. The $\mathtt{VP}_{\mathtt{PK}}$ specifies how much exposure time a circuit has against compromises of the ORs that use it. That is, until the $\mathtt{VP}_{\mathtt{PK}}$ elapses, the ORs have enough information to collectively decrypt circuit construction onions sent during that $\mathtt{VP}_{\mathtt{PK}}$. After each $\mathtt{VP}_{\mathtt{PK}}$, ORs discard their current private keys and obtain new keys from the PKGs. This period can be short, perhaps on the order of an hour.

The $\mathtt{VP}_{\mathtt{MK}}$ specifies the circuit's exposure time against compromises of the (distributed) PKG which reveal the master secret s. Because changing s involves the participation of all of the ORs as well as the PKGs, we suggest the $\mathtt{VP}_{\mathtt{MK}}$ be somewhat longer than the $\mathtt{VP}_{\mathtt{PK}}$, perhaps on the order of a day.[2] Remember that in the (n, t)-distributed PKG, if at least $n - t$ PKG members are honest and not compromised, no one will ever learn the value of a master secret.

Setup. Given the security parameter κ, the distributed PKGs generate a threshold signature key pair. Using the bootstrapping procedure described in Section 6.3.1, they also generate a prime p, groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ of order p and a bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. Finally, they choose a full-domain cryptographic hash function $\hat{H} : \{0,1\}^* \to \hat{\mathbb{G}}^*$. The distributed PKGs publish all of these values except its private signature key.

Key Generation. During the key generation step, the distributed PKGs perform following steps.

1. For each $\mathtt{VP}_{\mathtt{MK}}$, the distributed PKGs generate a random master key $s \in \mathbb{Z}_p^*$ and a random $g \in \mathbb{G}$, and calculates g^s using the distributed PKG protocol for BF-IBE (see Section 6.3.3). The PKG publishes a signed copy of (v_m, g, g^s), where v_m is a timestamp for the $\mathtt{VP}_{\mathtt{MK}}$ in question. This triple is to be shared by all users of the system.
2. For every valid OR with identity $\mathtt{OR}_i$, and for every $\mathtt{VP}_{\mathtt{PK}}$ v that overlaps with the $\mathtt{VP}_{\mathtt{MK}}$, the distributed PKGs generate the private key $d_{v,i} = \hat{H}(v \| \mathtt{OR}_i)^s$ using the distributed private-key extraction protocol (see Section 6.3.3), where $\|$ represents the usual concatenation operation.
3. The distributed PKGs distribute these private keys, as well as the signed copy of (v_m, g, g^s), to the appropriate ORs over a secure authenticated forward-secret channel. If an OR becomes compromised, the PKG can revoke it by simply no longer calculating its values of $d_{v,i}$.

[2]Note that $\mathtt{VP}_{\mathtt{MK}}$ and $\mathtt{VP}_{\mathtt{PK}}$ both are deployment parameters and do not affect the security analysis.

Note that this key distribution can be *batched*; that is, the distributed PKGs can precompute the private keys in advance (say a few hours at a time), and deliver them to the ORs in batches of any size from one $\mathtt{VP_{PK}}$ at a time on up. This batching reduces the amount of time the distributed PKGs have to be online, and does not sacrifice forward secrecy. On the other hand, large batches will delay the time until a revocation becomes effective.

User Setup. Once during each $\mathtt{VP_{MK}}$, every user has to obtain the signed triple (v_m, g, g^s) from any OR or from a public website. Once during each $\mathtt{VP_{PK}}$, every user has to compute the following pairing for each OR i and store the results locally:

$$\gamma_{v,i} = e(g^s, h_{v,i}) = e(g, h_{v,i})^s \text{ where } h_{v,i} = \hat{H}(v \| \mathtt{OR}_i) \ .$$

Circuit Construction. During a $\mathtt{VP_{PK}}$ v, a user U chooses ℓ ORs (say $\mathtt{OR}_1, \mathtt{OR}_2, \ldots, \mathtt{OR}_\ell$) and constructs a circuit $U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell$ with the following steps.

1. For each $\mathtt{OR}_i$ in the circuit, the user generates a random integer $r_i \in \mathbb{Z}_p^*$ and computes the pseudonym $\alpha_{Ui} = g^{r_i}$ and the value ${\gamma_{v,i}}^{r_i} = e(g, h_{v,i})^{sr_i}$. From ${\gamma_{v,i}}^{r_i}$ two session keys are derived: a forward session key $K_{U,i}$ and a backward session key $K_{i,U}$. Finally, the following onion is built and sent to $\mathtt{OR}_1$, the first OR in the circuit:

$$g^{r_1}, \{\mathtt{OR}_2, g^{r_2}, \{\cdots\{\mathtt{OR}_\ell, g^{r_\ell}, \{\emptyset\}_{K_{U,\ell}}\}\cdots\}_{K_{U,2}}\}_{K_{U,1}} \tag{7.1}$$

   Here $\{\cdots\}_{K_{U,i}}$ is symmetric-key encryption and $\emptyset$ is an empty message which informs $\mathtt{OR}_\ell$ that $\mathtt{OR}_\ell$ is the exit node.

2. After receiving the onion, the OR with identity $\mathtt{OR}_i$ uses the received g^{r_i} and its currently valid private key $d_{v,i}$ to compute $e(g^{r_i}, d_{v,i}) = e(g, h_i)^{r_i s} = {\gamma_{v,i}}^{r_i}$. It derives the forward session key $K_{U,i}$ and the backward session key $K_{i,U}$. It decrypts the outermost onion layer $\{\cdots\}_{K_{U,i}}$ to obtain the user's next pseudonym, the nested ciphertext, and the identity of the next node in the circuit. The OR then forwards the pseudonym and ciphertext to the next node. To avoid replay attacks, it also stores pseudonyms. The process ends when an OR ($\mathtt{OR}_\ell$ in this case) gets $\emptyset$.

3. The exit node $\mathtt{OR}_\ell$ sends a confirmation message encrypted with the backward session key $\{\text{Confirm}\}_{K_{\ell,U}}$ to the previous OR in the circuit. Each OR encrypts the confirmation with its backward session key and sends it to the previous node, until the ciphertext reaches the user. The user decrypts the ciphertext layers to verify the confirmation.

4. If the user does not receive the confirmation in a specified time, she selects a different set of ORs and repeats the protocol.

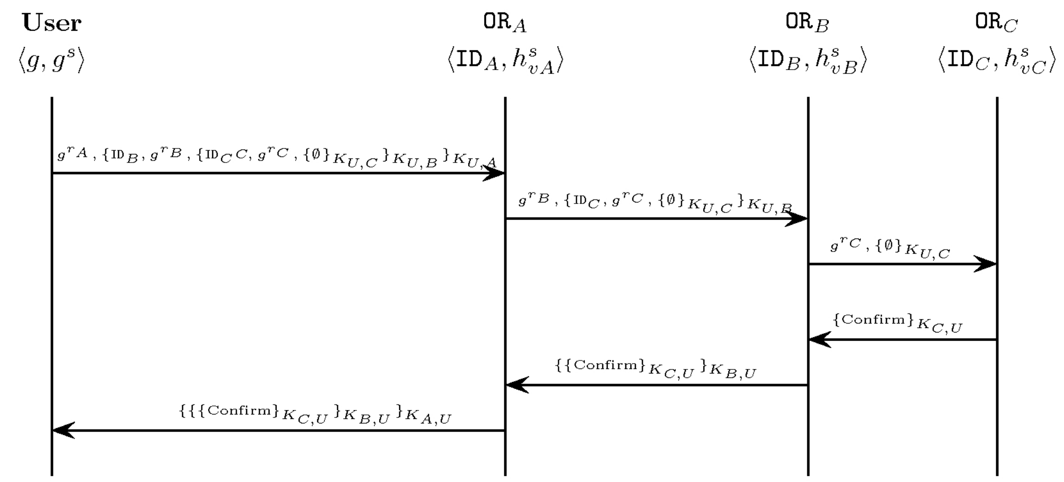
User
$\langle g, g^s \rangle$
$\mathtt{OR}_A$
$\langle \mathtt{ID}_A, h^s_{vA} \rangle$
$\mathtt{OR}_B$
$\langle \mathtt{ID}_B, h^s_{vB} \rangle$
$\mathtt{OR}_C$
$\langle \mathtt{ID}_C, h^s_{vC} \rangle$
$g^{r_A}, \{\mathrm{ID}_B, g^{r_B}, \{\mathrm{ID}_C C, g^{r_C}, \{\emptyset\}_{K_{U,C}}\}_{K_{U,B}}\}_{K_{U,A}}$
$g^{r_B}, \{\mathrm{ID}_C, g^{r_C}, \{\emptyset\}_{K_{U,C}}\}_{K_{U,B}}$
$g^{r_C}, \{\emptyset\}_{K_{U,C}}$
$\{\mathrm{Confirm}\}_{K_{C,U}}$
$\{\{\mathrm{Confirm}\}_{K_{C,U}}\}_{K_{B,U}}$
$\{\{\{\mathrm{Confirm}\}_{K_{C,U}}\}_{K_{B,U}}\}_{K_{A,U}}$

Figure 7.1: A user builds a circuit with three ORs.

The circuit construction is further illustrated in Figure 7.1, where a user builds a three-node circuit.

Anonymous Communication. After the circuit is constructed, communication proceeds in the same manner as in Tor. The user sends onions through the circuit with each layer encrypted with the forward keys $K_{U,i}$, and each hop decrypts one layer. Replies are encrypted at each hop with the backward key $K_{i,U}$, and the user decrypts the received onion.

Note that as an optimization, one or more messages can be bundled inside the original circuit construction onion, in place of $\emptyset$.

This PB-OR circuit construction provides forward secrecy only after ORs' private keys are rotated. In other words (as defined by Øverlier and Syverson [ØS07]), it only provides *eventual* forward secrecy rather than *immediate* forward secrecy. In the next section, we resolve this issue, without a significant increase in circuit construction time, by introducing a partially interactive onion routing circuit construction.

7.4 λ-pass Onion Routing

Tor achieves immediate forward secrecy using telescoping. Telescoping can also be considered as an ℓ-pass circuit construction, where ℓ is the circuit length, with immediate forward secrecy at each of the OR nodes. In practice however, it is sufficient to have immediate forward secrecy at fewer than ℓ nodes, as an adversary will be stymied when it encounters any such node. In this section, we define λ-pass onion routing circuit construction, which achieves immediate forward secrecy at λ nodes (for $2 < \lambda \leq \ell$) with reduced circuit construction cost over telescoping.

Impossibility of Immediate Forward Secrecy in Single-pass Circuit Construction. To motivate multiple pass circuit construction, we will describe why it is impossible to obtain immediate forward secrecy in any single-pass circuit construction, regardless of the cryptographic setting.

In an immediately forward secret circuit construction, compromise of OR private keys after a circuit is built should not allow any information about the circuit path to be recovered. Further, after the circuit is destroyed and the keys are dropped, it should not be possible for any honest user and an OR to re-compute their shared keys for that session. To achieve these properties, *both* parties must contribute some randomness to the creation of the the session key and they must drop these random values once the session keys are generated. Consequently, before the user can generate the forward session key, the random values (in some modified form) have to be exchanged between the user and the OR. The modified forms should enable only the authentic receiver to compute the session key. In an immediate forward secret circuit construction like Tor, these session-dependent random values are realized using the DH exponents (x, y), while the DH parameters (g^x, g^y) provide the publicly exchanged forms of the randomness.

In any single-pass circuit construction, an OR does not reply immediately after receiving an onion (except for exit nodes). Therefore, addition of any randomness from the OR in the forward session key is not possible, before that session key can be used to convey the OR its successor. Consequently, any time later in the same $\mathtt{VP}_{\mathtt{PK}}$, an adversary can compromise the OR, use the OR's private key to generate the session keys and exploit those to find the next node in the circuit path. Further, although it is possible for nodes to send their part of randomness for session keys along with the (backward) confirmation onion, this does not provide any advantage as the adversary can always find the circuit path by decrypting the forward onion. Thus we see that it is not possible to obtain immediate forward secrecy in a single-pass circuit construction.

λ-pass Circuit Construction Protocol. As replies from the last node of single-pass circuit constructions are direct, immediate forward secrecy is easy to achieve at this node. Here, we consider $\lambda - 1$ additional single-pass circuit constructions to achieve immediate forward secrecy at λ nodes. We note that Øverlier and Syverson [ØS07, Protocol 3] propose a similar circuit construction, but their focus is on dealing with replay attacks. Our λ-pass protocol adds message flows to provide partial immediate forward secrecy, whereas their protocol uses an increased number of flows to prevent replay of construction onions.

Although our λ-pass circuit construction can be applied in any public-key setting, for simplicity, here we present it for pairing-based onion routing, as defined in Section 7.3.

Setup, Key Generation, User Setup. Same as that of PB-OR in Section 7.3.

Circuit Construction. During a $\texttt{VP}_{\texttt{PK}}$ v, a user U chooses a set of ORs (say $\texttt{OR}_1, \texttt{OR}_2, \ldots, \texttt{OR}_\ell$) and constructs a circuit $U \Leftrightarrow \texttt{OR}_1 \Leftrightarrow \texttt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \texttt{OR}_\ell$ with the following steps.

1. The user selects λ indices $\Lambda_1 < \Lambda_2 < \cdots < \Lambda_{\lambda-1} < \Lambda_\lambda = \ell$.
2. As in Section 7.3, for each $\texttt{OR}_i$ in the circuit, the user generates a random integer $r_i \in \mathbb{Z}_p^*$ and computes the pseudonym $\alpha_{Ui} = g^{r_i}$ and the value ${\gamma_{v,i}}^{r_i} = e(g, h_{v,i})^{sr_i}$. From ${\gamma_{v,i}}^{r_i}$ two session keys are derived: a forward session key $K_{U,i}$ and a backward session key $K_{i,U}$. At this stage, the user erases the random values r_i for all but $r_{\Lambda_1}, r_{\Lambda_2}, \ldots, r_{\Lambda_\lambda}$.
3. The user then creates the following onion and sends it to $\texttt{OR}_1$.

$$g^{r_1}, \{\texttt{OR}_2, g^{r_2}, \{\cdots\{\texttt{OR}_{\Lambda_1}, g^{r_{\Lambda_1}}, \{\emptyset\}_{K_{U,\Lambda_1}}\}\cdots\}_{K_{U,2}}\}_{K_{U,1}}$$

   Here $\{\cdots\}_{K_{U,i}}$ is symmetric-key encryption and $\emptyset$ is an empty message which informs $\texttt{OR}_{\Lambda_1}$ that it is the last node.
4. Each node $\texttt{OR}_i$ with $i \geq 1$, uses g^{r_i} and its currently valid private key $d_{v,i}$ to compute $e(g^{r_i}, d_{v,i}) = e(g, h_i)^{r_i s} = {\gamma_{v,i}}^{r_i}$. It derives the forward session keys $K_{U,i}$ and the backward session keys $K_{i,U}$. It decrypts the outermost onion layer $\{\cdots\}_{K_{U,i}}$ to obtain the user's next pseudonym, the nested ciphertext, and the identity of the next node in the circuit. The OR then forwards the pseudonym and ciphertext to the next node. To avoid replay attacks, it also stores pseudonyms. The process ends when $\texttt{OR}_{\Lambda_1}$ gets $\emptyset$.
5. The last node in the partial circuit $\texttt{OR}_{\Lambda_1}$ generates a random integer $r_{U_1} \in \mathbb{Z}_p^*$, and computes a pseudonym $(h_{v,\Lambda_1})^{r_{U_1}}$. It then generates ${\gamma_{v,\Lambda_1}^{r_{\Lambda_1}}}^{r_{U_1}}$, derives modified forward and backward session keys (K^*_{U,Λ_1} and $K^*_{\Lambda_1,U}$) and sends a confirmation message encrypted with the backward session key $\{\text{Confirm}\}_{K^*_{\Lambda_1,U}}$ along with the pseudonym $(h_{v,\Lambda_1})^{r_{U_1}}$ to the previous OR in the circuit. To obtain immediate forward secrecy, it also erases the random integer r_{U_1} and the value ${\gamma_{v,\Lambda_1}^{r_{\Lambda_1}}}^{r_{U_1}}$ right away, and will erase K^*_{U,Λ_1} and $K^*_{\Lambda_1,U}$ immediately after the circuit is no longer in use.
6. Each OR encrypts the confirmation with its backward session key and sends it to the previous node, until the ciphertext reaches the user. The user decrypts the ciphertext layers to verify the confirmation and in the process, generates the modified session keys for $\texttt{OR}_{\Lambda_1}$ using the received pseudonym ${h_{v,\Lambda_1}}^{r_{U_1}}$ and the stored random value r_{Λ_1}. After this, the user drops the random value r_{Λ_1}.

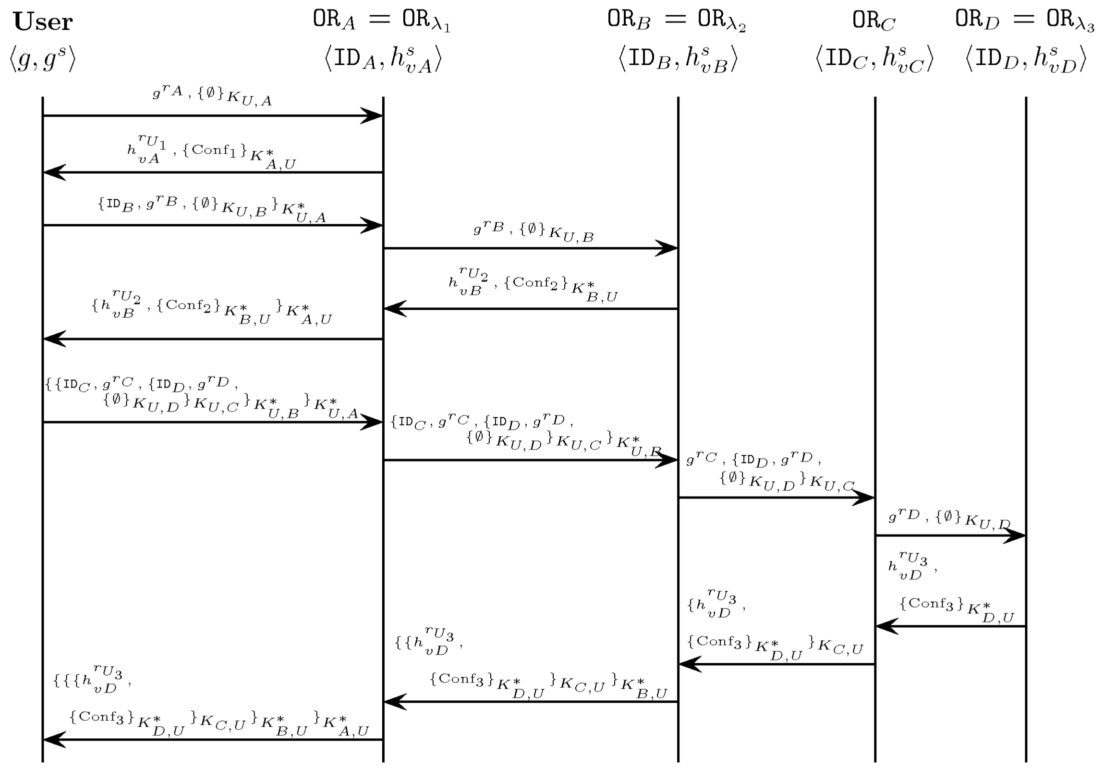
User
$\langle g, g^s \rangle$
$\mathtt{OR}_A = \mathtt{OR}_{\lambda_1}$
$\langle \mathtt{ID}_A, h^s_{vA} \rangle$
$\mathtt{OR}_B = \mathtt{OR}_{\lambda_2}$
$\langle \mathtt{ID}_B, h^s_{vB} \rangle$
$\mathtt{OR}_C$
$\langle \mathtt{ID}_C, h^s_{vC} \rangle$
$\mathtt{OR}_D = \mathtt{OR}_{\lambda_3}$
$\langle \mathtt{ID}_D, h^s_{vD} \rangle$
$g^{r_A}, \{\emptyset\}_{K_{U,A}}$
$h_{vA}^{r_{U_1}}, \{\mathrm{Conf}_1\}_{K^*_{A,U}}$
$\{\mathtt{ID}_B, g^{r_B}, \{\emptyset\}_{K_{U,B}}\}_{K^*_{U,A}}$
$g^{r_B}, \{\emptyset\}_{K_{U,B}}$
$h_{vB}^{r_{U_2}}, \{\mathrm{Conf}_2\}_{K^*_{B,U}}$
$\{h_{vB}^{r_{U_2}}, \{\mathrm{Conf}_2\}_{K^*_{B,U}}\}_{K^*_{A,U}}$
$\{\{\mathtt{ID}_C, g^{r_C}, \{\mathtt{ID}_D, g^{r_D}, \{\emptyset\}_{K_{U,D}}\}_{K_{U,C}}\}_{K^*_{U,B}}\}_{K^*_{U,A}}$
$\{\mathtt{ID}_C, g^{r_C}, \{\mathtt{ID}_D, g^{r_D}, \{\emptyset\}_{K_{U,D}}\}_{K_{U,C}}\}_{K^*_{U,B}}$
$g^{r_C}, \{\mathtt{ID}_D, g^{r_D}, \{\emptyset\}_{K_{U,D}}\}_{K_{U,C}}$
$g^{r_D}, \{\emptyset\}_{K_{U,D}}$
$h_{vD}^{r_{U_3}}, \{\mathrm{Conf}_3\}_{K^*_{D,U}}$
$\{h_{vD}^{r_{U_3}}, \{\mathrm{Conf}_3\}_{K^*_{D,U}}\}_{K_{C,U}}$
$\{\{h_{vD}^{r_{U_3}}, \{\mathrm{Conf}_3\}_{K^*_{D,U}}\}_{K_{C,U}}\}_{K^*_{B,U}}$
$\{\{\{h_{vD}^{r_{U_3}}, \{\mathrm{Conf}_3\}_{K^*_{D,U}}\}_{K_{C,U}}\}_{K^*_{B,U}}\}_{K^*_{A,U}}$

Figure 7.2: A user builds a circuit with four ORs and $\lambda = 3$.

7. Now, the partial circuit $U \Leftrightarrow A \cdots \Leftrightarrow \mathtt{OR}_{\Lambda_1}$ is used to extend the circuit to $\mathtt{OR}_{\Lambda_2}$ by sending the following onion to $\mathtt{OR}_1$.

$$\{\{\cdots\{\mathtt{OR}_{\Lambda_1+1}, g^{r_{\Lambda_1+1}}, \{\cdots\{\mathtt{OR}_{\Lambda_2}, g^{r_{\Lambda_2}}, \{\emptyset\}_{K_{U,\Lambda_2}}\}\cdots\}_{K_{U,\Lambda_1+1}}\}_{K^*_{U,\Lambda_1}}\cdots\}_{K_{U,2}}\}_{K_{U,1}}$$

8. The user completes λ passes to construct the complete circuit $U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell$.
9. If the user does not receive any of the λ confirmations in specified times, she selects a different set of ORs and repeats the protocol.

Anonymous Communication. Same as that of PB-OR in Section 7.3.

This circuit construction is further illustrated in Figure 7.2, where a user builds a four-node circuit with $\lambda = 3$.

Value of λ and node placement. For $\lambda = \ell$, the above circuit becomes a telescoping circuit construction. Øverlier and Syverson [ØS07] observe that, in Tor, there is always forward secrecy at the entry node, as the link between the user and the entry node in the circuit is encrypted using TLS. Therefore, our circuit construction defined in Section 7.3, without any significant modification, can easily be a 2-pass circuit construction having immediate forward secrecy at the entry node and the exit node.

Considering an adversary who controls some of the OR network (but not all, as in our threat model, Section 7.3), it is certainly advantageous to keep $2 < \lambda \leq \ell$. We observe that for Tor, with a network of more than a thousand nodes, assuming a non-global and non-adaptive adversary, with access to an infrequently changing small part of the network, it is sufficient to have $\lambda = 3$; that is, immediate forward secrecy at the entry node, exit node and one of the nodes in between. In this case, after the circuit is closed, the adversary's successive compromise of the ORs in a circuit is thwarted once it reaches the immediate forward secret node. In other words, assuming that the adversary has access to a few ORs in a circuit and can compromise all others in the network in the future, it still cannot link the two parts of the circuit divided at the immediate forward secret node. However, for a stronger adversary in a network with longer circuits, a larger value of λ may be required.

An immediate question is the placement of the immediate forward secret nodes in the circuit path. It is easy to observe that a circuit with two adjacent immediate forward secret nodes is more difficult to attack using traffic analysis than one where those two nodes are separated. Further, as the ultimate goal of onion routing is anonymity for the sender and receiver, it is good to have immediate forward secret nodes at the start and at the end of the circuit. Therefore, we suggest $\lceil \lambda/2 \rceil$ immediate forward secret nodes at the start of the circuit and remaining $\lfloor \lambda/2 \rfloor$ immediate forward secret nodes at the end of the circuit. In cases when the recipient does not require anonymity (e.g. it is a web server), the efficiency of the construction can be improved by placing the first $\lambda - 1$ nodes closest to the sender; that is, by selecting $\Lambda_i = i$ for $1 \leq i \leq \lambda - 1$ and $\Lambda_\lambda = \ell$. An attacker who observes the onion past the last forward secret node may be able to decrypt the remaining layers, but the first $\lambda - 1$ forward secret nodes have already provided anonymity for the sender.

Once the random parameters r_{U_i} and r_{Λ_i} are dropped by the $\mathtt{OR}_{\Lambda_i}$ and the user U respectively, deriving their session keys becomes the BDH problem, even if $\mathtt{OR}_{\Lambda_i}$ gets compromised during the $\mathtt{VP}_{\mathtt{PK}}$. Therefore, we achieve immediate forward secrecy at λ ORs and the $\mathtt{VP}_{\mathtt{PK}}$ could be longer, or made equal to the $\mathtt{VP}_{\mathtt{MK}}$. The latter would also eliminate the need to attach validity periods to OR identities.

7.5 Using Sphinx in PB-OR

Mix message formats have been a point of interest in research on mix networks [MCPS03, Möl03, DDM03, DL04, CL05, SSH08, DG09]. Recently, Danezis and Goldberg [DG09] proposed Sphinx as the most compact and efficient cryptographic mix message format and proved its security in the UC model. In this section, we present PB-OR circuit construction using the Sphinx methodology and discuss its security properties in the UC model.

The Sphinx Message Format. In Sphinx, an adversary is computationally bounded by a security parameter κ. Let $\mathbb{G}$ be a cyclic group of order p. Sphinx makes the circuit construction message size independent of the length of the circuit, ℓ; we denote the maximum length of a circuit as ν. Node identifiers are κ-bit strings. Each node has a public/private key pair. Further, Sphinx assumes a message authentication code (MAC) μ, a pseudo-random generator (PRG) ρ and corresponding random oracle hash functions $H_\mu, H_\rho : \mathbb{G}^* \to \{0,1\}^\kappa$. It also needs a random oracle hash function $H_b : \mathbb{G}^* \times \mathbb{G}^* \to \mathbb{Z}_p^*$.

Cryptographically, the most elegant feature of the Sphinx message format is its session key derivation technique based on a repeatedly modified random element of $\mathbb{G}$. We call this technique Sphinx's *blinding logic*. The mentioned random element (α) and its repeated modified forms are called *pseudonyms* since each of these random elements is a temporary public key whose private key is held by the user. In the Sphinx blinding logic, each mix node uses a pseudonym supplied by its predecessor and its own private key to compute the session key with the user. To provide unlinkability, a pseudonym must not remain the same across the circuit. In the onion routing literature, this is done by including separate random pseudonyms in a construction message for each node in the circuit. In Sphinx's blinding logic, this is achieved using a single repeatedly changing pseudonym. At every node, a blinding factor is extracted from the current pseudonym α_i and the newly computed session key s_i. The pseudonym is then exponentiated with the blinding factor to generate the next pseudonym. In other words,

$$\alpha_{i+1} = \alpha_i^{H_b(\alpha_i, s_i)}. \tag{7.2}$$

The session key s_i in Sphinx is computed as in the half-certificated Diffie-Hellman key agreement [MOV97, Section 12.6]; we will soon see that our construction in the identity-based setting is different.

To send an anonymous message, a sender first chooses her mix nodes and obtains their public keys. She then computes α_i and s_i and wraps the message in multiple layers of encryption using the PRG ρ to generate ciphertext values β_i. To check the integrity of the message header, she calculates and includes a MAC γ_i at each mixing stage. Upon receiving a message header $(\alpha_i, \beta_i, \gamma_i)$, each mix node $\mathtt{ID}_i$ extracts session keys using its private key

x_i and the pseudonym α_i received from the predecessor. It uses those to verify the MAC γ_i and to decrypt a layer of encryption of β_i. It also extracts the routing information, computes the pseudonym α_{i+1} for the next node using Equation (7.2) and forwards the message to $\mathtt{ID}_{i+1}$. As we are working towards using Sphinx in OR circuit construction, we ignore Sphinx's payloads and reply blocks.

Note that we are here concerned only with the circuit construction messages. Our methodologies are not required for OR communications in already constructed circuits.

Recent Enhancements in OR Circuit Construction. Along with already discussed Tor-preDH and PB-OR, more recently Catalano, Fiore and Gennaro [CFG09] suggested improvements to Tor circuit construction. These schemes use a *one-way anonymous* key agreement [KZG07] strategy in the PKC, IBC and certificateless cryptography (CLC) settings respectively. Here, a user chooses a random element of $\mathbb{Z}_p^*$ per circuit node and computes an associated pseudonym. A session key is computed using the node's public key and the random element at the user end, and using the pseudonym received and the node's private key at the node's end; the precise session-key computation and the cryptographic assumption vary with the OR circuit construction protocol. Most importantly, unlike the Tor authentication protocol (TAP), the user does not encrypt the pseudonyms in these schemes, which is a direct result of the inclusion of the private key of an OR node in the session key generation. Therefore, it is possible to incorporate Sphinx's blinding logic into these schemes. In this section, we concentrate on using Sphinx's Blinding logic in PB-OR.

Using Sphinx in PB-OR. In PB-OR, a user generates a random $r_i \in_R \mathbb{Z}_p^*$ and sends a pseudonym $\alpha_i = g^{r_i}$ to node $\mathtt{OR}_i$ over the already-formed circuit (if any). The session key s_i is generated at the user end as $s_i = e(y, \hat{H}(\mathtt{ID}_i))^{r_i}$ and at the node $\mathtt{OR}_i$ as $s_i = e(\alpha_i, d_i)$.

For the Sphinx-based PB-OR circuit construction, we slightly modify the Sphinx notation. In the original Sphinx format, a destination address (Δ) and a reply block identifier $I \in \{0,1\}^\kappa$ are present. Since we do not require circuit construction messages to be delivered to external parties, we can remove these portions of the Sphinx format, saving 2κ bits in the size of β_i. As the sets to which the session keys s_i belong changes to $\mathbb{G}_T$ for PB-OR, we also modify the definition of our hash functions as follows: $H_\mu, H_\rho : \mathbb{G}_T \to \{0,1\}^\kappa$ and $H_b : \mathbb{G}^* \times \mathbb{G}_T \to \mathbb{Z}_p^*$.

To create a PB-OR circuit construction message, we use Sphinx's mix header creation algorithm ([DG09, Section 3.2]) with a modification to the session key generation. The original Sphinx message format is based on the half-certified DH key agreement [MOV97, Section 12.6], where a session key s_i is generated as $s_i = y_i^{x b_0 b_1 \cdots b_{i-1}}$ at the user's end and as $s_i = \alpha_i^{x_i}$ at node $\mathtt{OR}_i$, where (y_i, x_i) is the public/private key pair for $\mathtt{OR}_i$, α_i is a pseudonym for node $\mathtt{OR}_i$, x is the session-specific randomness and $b_0, \ldots, b_{i-1}$ are the blinding factors, $b_i = H_b(\alpha_i, s_i)$. For our PB-OR design using Sphinx, α_i can be generated as $\alpha_i = \alpha_{i-1}^{H_b(\alpha_{i-1}, s_{i-1})}$, while the computation of s_i remains the same as that of the original PB-OR, except here $r_i = xb_0b_1 \cdots b_{i-1}$ for an $x \in_R \mathbb{Z}_p^*$ chosen by the user.

Note that, although Sphinx is defined for single-pass constructions, its blinding logic is also useful in multi-pass constructions, where it can avoid the transfer of pseudonyms in circuit extension messages. However, here, we concentrate on the single-pass version.

Security Analysis. Camenisch and Lysyanskaya [CL05] design a framework for onion routing with provable security in the UC model. They define onion-correctness, onion-integrity and onion-security properties for an OR scheme and prove Theorem 7.2.

Theorem 7.2 (Theorem 1 [CL05])**.** *An onion routing scheme satisfying onion-correctness, integrity and security, when combined with secure point-to-point secure channels, yields a UC-secure OR scheme.*

Danezis and Goldberg [DG09] separate a wrap-resistance property from onion-security to simplify the onion-security definition and prove the resulting four security properties of the Sphinx message format using random oracles. We use their security discussion to define the security requirements for our PB-OR circuit design.

Onion-correctness. According to [CL05], an OR circuit construction is *correct* if a message reaches the intended recipient in a constructed circuit whenever an onion is formed correctly, processed by the right routers in the right order, and these routers follow the protocol. It is easy to observe that a Sphinx-based PB-OR circuit construction works correctly.

Onion-integrity. An OR circuit construction has *integrity* if it is not possible for an adversary to build a circuit of more than N honest nodes, for some predefined bound N, except with negligible probability. As we only change the session key generation step in the original Sphinx message format, our proof of integrity remains exactly the same as that of Sphinx in [DG09, Sec. 4.2] except in our case $N = \nu$ instead of $\nu + 1$ as we reduce the size of β_i by 2κ bits.

Wrap-resistance. Camenisch and Lysyanskaya [CL05] also suggest a security property such that it should not be possible for an OR node to wrap an existing onion; that is, given a target onion (say) O_{i+1}, it must not be possible for an adversary to construct onion O_i such that a node $\texttt{OR}_i$ processing O_i would output O_{i+1}. This must remain true even if the adversary can select $\texttt{OR}_i$'s private key. In Sphinx, the wrap-resistance property is based on the difficulty of computing α_i given $\alpha_{i+1} = \alpha_i^{H_b(\alpha_i, s_i)}$. As this blinding logic remains unchanged in our design, our wrap-resistance discussion is same as that of [DG09, Sec. 4.3].

Onion-security. As defined by [CL05], the *onion-security* property requires that an attacker controlling all but one honest node (say $\texttt{OR}^*$) in a circuit should not be able to distinguish OR circuit construction messages entering into the unattacked node $\texttt{OR}^*$. Onion-security for our construction differs in three places from the original Sphinx format.

1. There are no reply messages and there is no need for indistinguishability of forward and reply messages.
2. There are no destination addresses Δ and no message payloads.
3. Generation of the session key varies with the circuit construction protocol.

We observe that the first two differences do not affect the security proof. The third difference is related to the infeasibility of the adversaries to distinguish between s_i generated while creating circuit and a random $s_i \in_R \mathbb{G}_T$. In PB-OR with Sphinx, except with negligible probability, it should not be feasible for an adversary to distinguish $s_i = e(g, \hat{H}(\texttt{ID}_i))^{sr_i} \in \mathbb{G}_T$ from a random element of $\mathbb{G}_T$, given g^s, $\alpha_i = g^{r_i}$ and $\hat{H}(\texttt{ID}_i) \in \hat{\mathbb{G}}$. For the bilinear pairing tuple $(e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$, given an instance of the DBDH problem $\langle g, g^a, g^b, \hat{g}, \hat{g}^a, \hat{g}^c, z : z \overset{?}{=} e(g, \hat{g})^{abc} \rangle$, it can be modelled as the above indistinguishability game using the following mapping: $\alpha_i = g^a$, $g^s = g^b$ and $\hat{H}(\texttt{ID}_i) = \hat{g}^c$. Therefore, the onion security property is achieved under the DBDH assumption for a tuple $(e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$.

7.6 Performance Comparison

In this section, we consider the cost of single pass and λ-pass circuit constructions from a user through ℓ onion routers. We estimate the computational cost, and count the number of AES-encrypted network communications. We compare the performance of our systems to that of Tor. We also analysis the performance of the Sphinx-based circuit construction in terms of message size and computational cost.

7.6.1 Performance of PB-OR and λ-pass PB-OR

Security Levels and Parameter Sizes. Before comparing the costs of the cryptography in the schemes we determine the parameter sizes required to provide the same level of security currently provided by Tor.

Tor uses public key parameters to provide security at the 80-bit level [Gol06]. The discrete log problem is in a 1024-bit field, and the RSA problem uses a 1024-bit modulus.

The symmetric parameters provide significantly more security, by using AES with a 128-bit key. We must choose appropriate groups $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\mathbb{G}_T$ over which our pairing will be defined in order to offer similar strength. For simplicity, we work with the symmetric pairing ($\mathbb{G} = \hat{\mathbb{G}}$); performance of PB-OR will only improve if we use the pairing of types 2 and 3. The current favourite choice is the group of torsion points of an elliptic curve group over a finite field, with either the Weil or Tate pairing. To achieve an 80-bit security level, the elliptic curve discrete log problem an attacker faces must be in a group of at least 160 bits. Due to the reduction of Menezes, Okamoto and Vanstone [MOV91], we must also ensure that discrete logs are intractable in the target group, $\mathbb{G}_T$. In our case, $\mathbb{G}_T = \mathbb{F}_{p^k}$, where k is the embedding degree of our curve taken over $\mathbb{F}_p$. We must then choose our curve E, a prime p, and embedding degree k such that $E(\mathbb{F}_p)$ has a cyclic subgroup of prime order $n \approx 2^{160}$, and p^k is around 2^{1024}. This can be achieved in a variety of ways, but two common choices are $k = 2, p \approx 2^{512}$ and $k = 6, p \approx 2^{171}$. Pairing implementations with both sets of parameters are available in the PBC library [Lyn09]. Efficiency studies suggest that $k = 2$ and the Tate pairing can offer better performance at this security level [KM05], so we make that choice.

Cost of Building a Circuit with Tor. Tor builds circuits by telescoping. A user Uriel chooses a Tor node (say Alice), and establishes a secure channel using an encrypted DH exchange. She then picks a second node, Bob, and over this secure channel, establishes a new secure channel to Bob with another (end-to-end) encrypted DH exchange. She proceeds in this manner until the circuit is of some desired length ℓ. For details, see the Tor specification [DM08]. Note that Uriel cannot use the same DH parameters with different nodes, lest those nodes be able to determine that the same user was communicating with each of them.

Each DH exchange requires Uriel to perform two modular exponentiations with 1024-bit moduli and 320-bit exponents. Likewise, each server also performs two of these exponentiations. Uriel RSA-encrypts the DH parameter she sends to the server, and the server decrypts it. The AES and hashing operations involved have negligible costs compared to these.

Uriel's circuit construction to Alice takes two messages: one from Uriel to Alice, and one from Alice to Uriel. When Uriel extends this circuit to Bob (via Alice), there are four additional messages: Uriel to Alice, Alice to Bob, Bob to Alice, and Alice to Uriel. Continuing in this way, we see that the total number of messages required for Tor to construct a circuit of length ℓ is $\ell(\ell + 1)$. Note that each of these messages needs to be encrypted and decrypted at each hop.

Cost of Building a Circuit with Pairing-Based Onion Routing. In order to create a circuit of length ℓ with our single-pass circuit construction, the user Uriel must choose

Operation	Time	Tor		PB-OR		λ-Pass PB-OR		
	(ms)	user	OR	user	OR	user	efs-OR	ifs-OR
Pairing	2.9	0	0	0	1	λ	1	1
RSA decryption	2.7	0	1	0	0	0	0	0
Exponentiation (Tor)	1.5	2ℓ	2	0	0	0	0	0
Exponentiation in $\mathbb{G}$	1.0	0	0	ℓ	0	ℓ	0	1
Exponentiation in $\mathbb{G}_T$	0.2	0	0	ℓ	0	ℓ	0	1
RSA encryption	0.1	ℓ	0	0	0	0	0	0
Total time (ms)		3.1ℓ	5.7	1.2ℓ	2.9	$1.2\ell + 2.9\lambda$	2.9	4.1
Total AES-encrypted messages		$\ell(\ell + 1)$		2ℓ		$2\lambda\Lambda_{\text{avg}}$		

Table 7.1: Comparison of costs of setting up a circuit of length ℓ. The values in the Tor column are based on the Tor specification [DM08]. PB-OR represents our pairing-based onion routing schemes. efs-OR indicates eventual forward secret ORs, while ifs-OR indicates immediate forward secret ones. Λ_{avg} represents the average of the indices of the λ immediate forward secret nodes.

ℓ random elements r_i of $\mathbb{Z}_p^*$. As above, Uriel should not reuse these values. She then computes g^{r_S} and ${\gamma_S}^{r_S}$, and derives the forward and backward keys $K_{U,S}$ and $K_{S,U}$ from ${\gamma_S}^{r_S}$, for each server S in the circuit. Note that the γ_S values were precomputed, and cost nothing during each circuit creation. Each server computes $e(g^{r_S}, d_S) = {\gamma_S}^{r_S}$ for its current private key d_S and derives $K_{U,S}$ and $K_{S,U}$.

Uriel creates one message, as in Figure 7.1, and sends it to the first server in the chain. This server decrypts a layer and sends the result to the second server in the chain, and so on, for a total of ℓ hop-by-hop encrypted messages. At the end of the chain, the last server replies with a confirmation message that travels back through the chain, producing ℓ more messages, for a total of 2ℓ.

Cost of Building a λ-Pass Pairing-Based Onion Routing Circuit. Our λ-pass pairing-based onion routing circuit construction is similar to that of our single-pass construction. Additional tasks that the immediate forward secret nodes must do are generation of a random integer r_{U_i}, computation of the pseudonym $H_{v\Lambda_i}^{r_{U_i}}$, and computation of ${\gamma_{v\Lambda_i}^{r_{\Lambda_i}}}^{r_{U_i}}$. Uriel correspondingly has to perform λ pairing computations to generate modified session keys using the received pseudonyms ${H_{v\Lambda_i}}^{r_{U_i}}$ from λ immediate forward secret nodes. The number of messages and corresponding AES encryptions depends on the positions of the λ immediate forward secret nodes in the circuit. It is equal to $2\sum_{i=1}^{\lambda} \Lambda_i = 2\lambda\Lambda_{\text{avg}}$, where Λ_{avg} is the average of the indices of the immediate forward secret nodes in the circuit.

Comparison and Discussion We summarize the results of the previous three sections in Table 7.1. We count the number of “bignum” operations for each of the client and the servers, both for Tor and for our pairing-based onion routing protocols. We ignore the comparatively negligible computational costs of AES operations and hashing. For each bignum operation, we include a benchmark timing. These timings were gathered on a 3.0 GHz Pentium D desktop using the PBC pairing-based cryptography library (version 0.4.7) [Lyn09].

We can see that the total computation time to construct a circuit of length ℓ using our single-pass method is 61% less on the user side and 49% less on the OR side as compared to using Tor. In addition, this circuit construction uses only a linear number of AES-encrypted messages, while Tor uses a quadratic number. As compared to single-pass circuit construction, our λ-pass circuit construction requires an additional λ pairing computations by the user, requiring a total of $1.2\ell + 2.9\lambda$ ms, and on average $2.9 + 1.2\lambda/\ell$ ms for each of the ORs. For proposed values of $\lambda = 3, 4$, or 5, these are certainly reasonable times, considering the advantage of immediate forward secrecy, and having $\mathtt{VP}_{\mathtt{PK}} = \mathtt{VP}_{\mathtt{MK}}$.

7.6.2 Message Compactness using Sphinx

Along with the UC model security, message compactness is an important advantage of using Sphinx. It is easy to observe that the major savings in the length of a circuit construction message comes from reuse of a pseudonym to which blinding is added at each circuit node.

To mitigate a recent attack on Tor by Evans, Dingledine and Grothoff [EDG09], which uses long circuit paths that loop back, the maximum circuit length for recent versions of Tor is set as 8. Therefore, we set $\nu = 8$ for our Sphinx-based design. However, while comparing, we give an advantage to the original PB-OR protocol by using Tor's default circuit size $\ell = 3$ for them; using $\nu = 3$ in our design will only increase our advantage.

In the Sphinx-based PB-OR construction, the user sends the tuple $(\alpha_1, \beta_1, \gamma_1)$ to node $\mathtt{OR}_1$. The lengths of the elements in this tuple are p, $(2\nu - 1)\kappa$ and κ respectively. The total length, therefore, is equal to $p + 2\nu\kappa$. In the chosen ECC setting ($\kappa = 80, p = 512$), this is equal to $512 + 2 * 8 * 80 = 1792$ bits. In the original PB-OR protocols, this cost is equal to $\nu(p + 2\kappa)$ as each layer of onion in those constructions requires p bits for a pseudonym, κ bits for identity of the nodes and κ bits for message integrity. With $\kappa = 80$, $p = 512$ and $\ell = 3$, this length is equal to 2016 bits, which is larger than the message size in our Sphinx-based format that can make circuits of any length up to 8. For $\nu = 3$, the difference will be significant.

Although Sphinx achieves partial independence from the circuit size ν with a single pseudonym, ν is still present in the message-size expression due to node addresses and the integrity mechanism. In onion routing, a user should be able to choose its nodes from the available pool in an arbitrary fashion [SB08, MW08], so information theoretically, it is impossible to make the message size independent of ν.

Increase in the Computational Cost. Compact messages and security in the UC model do not come without some additional computational cost. However, importantly, there is no addition to the computations done by users (possibly hundreds of thousands of them), while the increase is easily manageable for OR nodes.

The computation at a user end remains the same except for a few additional low-level operations such as a finite field multiplication, a pseudorandom number generation and a few hashes having computational costs in μs. Each node in a circuit has to perform an additional exponentiation in $\mathbb{G}$ as it prepares the pseudonym for the next node. However, timing values computed using the pairing-based cryptography (PBC) library [Lyn09] indicate that one exponentiation in $\mathbb{G}$ costs around 1 ms on a desktop machine. This does not affect the overall circuit construction cost in practice, which is in seconds due to the network latency.

Part III

Implementation

Chapter 8

Distributed Key Generation Implementation over PlanetLab

We implemented our HybridDKG protocol from Chapter 4 and analyzed its performance over the PlanetLab platform [PACR03]. In this chapter, we discuss our implementation and experiments along with other system aspects of distributed key generation. We observed HybridDKG to be practical for use over the Internet, which is further illustrated in the next chapter, where we use HybridDKG for robust communication over DHTs.

In Section 8.1, we briefly describe the design and implementation of HybridDKG. In Section 8.2, we analyze the results from our experiments over PlanetLab and discuss resilience against denial-of-service (DoS) attacks and Sybil attacks. In Section 8.3, we propose some system-level optimizations for the HybridDKG protocol based on our analysis. These optimizations improve the performance of the protocols, without hampering its liveness or safety, when a leader behaves honestly and delays in the system remain within reasonable limits.

8.1 Software Design and Implementation

We design our DKG nodes as state machines (using the state machine replication approach [Lam78, Sch90]), where nodes move from one state to another based on the messages received. These messages are categorized into three types: operator messages, network messages and timer messages. The operator messages define interactions between nodes and their operators and are of two types: in and out. In an in message, an operator provides some input to the protocol, while an out message presents the protocol results to the operator. The network messages realize the protocol flow between nodes. Almost all messages in the HybridVSS and HybridDKG pseudocode in Figures 4.1, 4.2, 4.3 and 4.4 are of this type. Finally, the timer messages implement the weak synchrony assumption described in Section 4.2.1 in the form of start timer, stop timer and timeout. Our node will be in one of the following seven states: leader_unconfirmed, under_recovery, functional, agreement_started, agreement_completed, leader_change_started and dkg_completed. leader_unconfirmed is a starting phase, which indicates that a node does not have a sufficient number of lead-ch signatures to confirm a new leader. We also differentiate between agreement_completed and dkg_completed as a node may complete the broadcast by the leader before it completes the associated VSS instances. The rest of the states (under_recovery, functional, agreement_started, dkg_completed, and leader_change_started) have their apparent meanings.

We aim at building the distributed PKGs protocols that we defined in Chapter 6 and required for the PB-OR protocol in Chapter 7. Therefore, we consider a DKG implementation over pairing-friendly elliptic curves. We develop our object-oriented C++ implementation over 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 confidentiality and message authentication.[1] However, our DKG code is generic and can easily be modified to work with any other C/C++ number theoretic library. When using a different library, a C++ interface layer will have to be developed over that library, which will provide a cyclic group interface in the form of a C++ class as required for our polynomials, commitments and shares.

Our implementation replicates our HybridVSS and HybridDKG pseudocode and therefore has an event- (or message-) driven architecture. The similarity between the code and the pseudocode is intentional; it helped identify several errors in the code and omissions in the pseudocode. DKG nodes are single-threaded and the code is structured as a set of event handlers. This set contains a handler for each operator and network message, and a handler for each timer. Each handler corresponds to an input action and there are also methods that correspond to the internal actions in the system. The event handling loop works as follows: nodes wait in a select call for a network message to arrive, for an operator instruction or for a timer deadline to be reached and then they call the appropriate handler. The handler performs computations similar to the corresponding action in the pseudocode and then it invokes any methods corresponding to internal actions whose preconditions have become true. In most of the cases, it results in sending a message to an operator or over the network. Each message has a 3-byte generic header, which contains a tag that identifies the message type (1 byte) and the total size of the rest of the message (2 bytes). The structure of the message bodies varies from message type to message type.

[1]Note that *nodes* have TLS PKI certificates, which does not conflict with the goal of providing IBE private keys to *clients* in distributed PKGs.

8.2 Performance Analysis

Experimental Setup and Testing. In order to examine its realistic performance, we test our DKG implementation on the PlanetLab platform.

A typical PlanetLab machine configuration is 4×2.4 GHz cores with 4 GB RAM and 500 GB hard disk. [Pla07] In terms of network capacity, the average bandwidth between PlanetLab nodes is 64 Mbps [LSB$^+$05]. For our experiments, we chose the required number of PlanetLab nodes randomly from nodes having near-average configuration and bandwidth, and a reasonable liveness history. In terms of the geographical distribution, although our selections were baised towards nodes in Europe and America, we had a significant number (around 20 percent) of nodes choosen from the other continents. Taking advantage of the uniform operating system distribution and configuration over all PlanetLab nodes, we compiled and statically linked our code on a single node and replicated the executable over the rest. Note that we did not consider the loads of machines while selecting our nodes; those loads were unpredictable and varied a lot during our experiments. In order to determine an average performance, we ran the experiments at least three times for each parameter set.

During our tests, we studied the following aspects: possible sizes of the system, the average completion time of the protocol and the applicability of the weak synchrony assumption from Section 4.2.1 that we make for HybridDKG. Our HybridDKG protocol handles Byzantine attacks and the performance of the implementation against these malicious attacks should have been verified during the testing. However, in our HybridDKG analyses in Section 4.4.2, we observe that a t-limited Byzantine adversary cannot launch any attack other than delaying the network messages; this is a direct effect of working in the computational security setting. The verification mechanism in our HybridDKG and HybridVSS protocols can easily catch any modification to network messages, commitments or shares and messages that fail verification are dropped by the honest nodes without any further processing. As a result, we do not include any special Byzantine adversary code in our tests. Note that we still need to follow the security threshold t for the shared polynomials as otherwise the adversary can break the secrecy of the protocol.

Evaluation and Analysis. Based on our experiments over the PlanetLab platform, we make the following observations:

- We test the performance of our DKG implementation for systems of up to 40 nodes and measure the average HybridDKG completion time. Figure 8.1 presents our results in a graphical form. The values range from few seconds for 5 nodes to a little over an hour for 40 nodes. Further, we observe an approximately cubic growth in the average completion time. With the cubic complexity and the average completion time of more than an hour for 40 nodes, we observe that DKGs for larger distributed systems ($n > 50$) are not practical for the Internet. We kept the t and f values as close as possible in our experiments in this section, each around $(n-1)/5$.

- In Figure 8.1, we also measure minimum and maximum completion times for the experiments. The large gaps between those values demonstrate the robustness of the DKG system against the Internet's asynchronous nature and varied resource levels of the PlanetLab nodes that we chose.

- To check the applicability of the weak synchrony assumption [CL02] that we use in HybridDKG, we also tested the system with crashed leaders. In such scenarios, the DKG protocol successfully completed after a few leader changes. However, we observe that the average completion time of a system critically varies with the choice of $delay(T)$ function.

  During our experiments, we observed that, for $delay(T) = T$, the leader change takes significantly more time than that required to reliably broadcast the chosen instances. As a result, when a leader is crashed, most of the other nodes end up waiting for long periods even though the next possible leader is available. Further, this waiting period grows significantly in the case of multiple crashes.

  We observed that $delay(T) = T/\delta$ for $\delta = 2$ or 3 is a better choice in terms of a compromise between allowing an honest leader enough time to complete its broadcast and reducing the waiting period between leader changes. However, an appropriate $delay(T)$ function may change as the system parameters vary and we suggest that $delay(T)$ should only be finalized for a system after some rigorous testing.

- Further, we compared the (CPU) execution time for nodes against the time they spend on network transmission or waiting for other nodes. We observe that the the protocol execution time per node is significantly smaller than their completion periods (Table 8.1). This proves that the completion time periods are larger not because of the required computation; they are high rather due to network delays and will drop significantly if a more reliable network with better bandwidth (*e.g.*, an internal network in an organization or a cloud computing environment) is provided.

Defence against DoS and Sybil Attacks. The distributed nature of HybridDKG provides an inherent protection against DoS attacks and the inclusion of the crashed-node 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

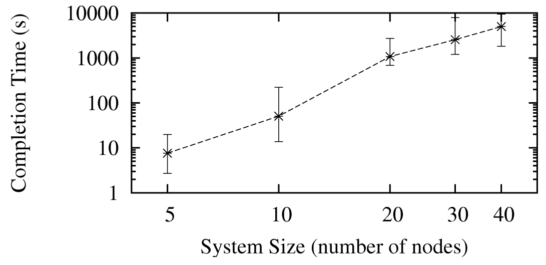

Figure 8.1: Completion time (with min/max bars) vs system size (log-log plot) for HybridDKG

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.

8.3 System-Level Optimizations

The completion time values of our HybridDKG implementation that we observe in Figure 8.1 are practical for applications such as PB-OR, where DKG phase sizes are in days. However, these values are not sufficient for many other practical systems such as our robust communication protocols for DHTs in the next chapter. During our experiments we observed that some system-level optimizations can significantly reduce the completion time values and make HybridDKG practical enough for these applications. We next discuss these optimizations. Note that these optimizations make HybridDKG more practical in the normal course of operation, when a leader behaves honestly and its messages flow without any significant delay. They may not the best in the worst-case scenarios having multiple leader-change operations. However, they never hamper the safety and liveness properties

Table 8.1: Median values of HybridDKG completion time and CPU time per node for various n values

n	t	f	**Completion Time** (sec)	**CPU Seconds/Node**
10	1	3	5.73	0.76
15	2	4	18.0	1.94
20	2	6	68.0	2.55
25	3	7	290.9	6.13
30	3	10	336.7	7.27

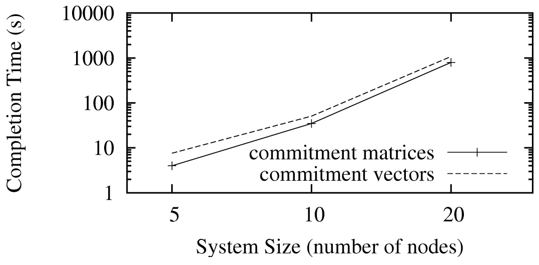

Figure 8.2: Completion time vs system size (log-log plot) for commitment matrices and commitment vectors

of the protocol.

shared Messages. With the PKI infrastructure in place, digital signatures are readily available in our system. Our HybridVSS scheme does not make use of these signatures. In the HybridDKG protocol, it is important for the leader to be one of first nodes to complete any HybridVSS instance. That helps the leader to send its DKG send message before the timer timeout*s* at fast nodes.

To help the leader, we add a new shared network message that a node having $2t+f+1$ signed VSS ready messages for a completed HybridVSS instance sends to a leader. The leader can then include this HybridVSS instance in its DKG send without completion of the VSS instance at its own machine. Note that $2t+f+1$ signed ready messages confirms that the corresponding HybridVSS instance will complete at all honest finally up nodes.

Commitment Matrices versus Commitment Vectors. In theory, linear size commitment vectors which use collision-resistant hash functions as introduced in [CKAS02] for AVSS, provide linear speedup over quadratic size commitment matrices. However, measuring precisely, the commitment size is $(t+1)^2$ for matrices and (approximately) $2(3t+2f+1)$ for vectors. Even if $f=0$, the required elements and computations for the matrix commitments are less than those for the vector commitments for $t<6$. For $f>0$, this upper limit will only increase.

Our experiments confirm this computation. We observed that the commitment matrices, although asymptotically inefficient, are more efficient in systems of size less than 20 (see Figure 8.2). We suggest the usage of commitment matrices instead of commitment vectors for systems of size less than 20.

Running only $2t+f+1$ VSS instances. We observed that the VSS instances are more resource consuming than the agreement required at the end. Generally, we only need $t+1$ VSS instances to succeed. Assuming $t+f$ VSS instances might fail during a DKG, it is sufficient to start HybridVSS instances at just $2t + f + 1$ nodes instead of at all n nodes. Nodes that do not start a VSS initially may utilize the weak synchrony assumption to determine when to start a VSS instance if required. Comparing the DKG completion time values in Figure 8.1 and Table 8.1, we observe that this optimization significantly reduces the completion time for HybridDKG.

We utilize these system optimizations in the next chapter, where we use our HybridDKG protocol to implement threshold signatures to achieve two robust communication protocols in peer-to-peer systems.

Chapter 9

Robust Communication in DHTs

9.1 Preliminaries

The peer-to-peer (P2P) paradigm is a popular approach to providing large-scale decentralized services. However, the lack of admission control in many such systems makes them vulnerable to malicious interference [SM02, Wal02]. This is a practical concern since large-scale P2P systems are in existence today such as the Azureus DHT [FPJ$^+$07] and the KAD DHT [SENB07], each of which see more than one million users per day. In addition to file sharing, there are proposals for using P2P systems to protect archived data [GKLL09], mitigate the impact of computer worms [ARS07] and re-implement the Domain Name System [PMTZ06]; such applications would likely benefit from increased security.

There are a number of results on peer-to-peer (P2P) systems that can provably tolerate Byzantine faults [AS06b, AS07, AS06a, SY08, FSY05, NW03, HK04, JAvR06]. To date, the majority of results pertain to distributed hash tables (DHTs); these DHTs are called *robust*. A common technique in robust DHTs is the use of *quorums*, which are sets of peers such that a minority of the members suffer adversarial faults. A quorum replaces an individual peer as the atomic unit and adversarial behaviour can be overcome by majority action allowing for communication between correct peers; we call this *robust communication*. Since critical operations such as data queries are performed in concert by members of a quorum, robust communication must be efficient.

Several protocols using quorums have been proposed; however, there is a common theme in the way such quorums are utilized. A message m originating from a peer P traverses a sequence of quorums Q_1, Q_2, $\ldots$, Q_ℓ until a destination peer is reached. A typical example is a query for content where the destination is a peer Q holding a data item. Initially P notifies its own quorum Q_1 that it wishes to transmit m. Each peer in Q_1 forwards m to all peers in Q_2. A peer in Q_2 determines the correct message by majority filtering on all incoming messages and, in turn, sends to all peers in the next quorum. This forwarding process continues until the quorum Q_ℓ holding Q is reached. Assuming a majority of correct peers in each quorum, transmission of m is guaranteed. Unfortunately, this simple protocol is costly. If all quorums have size n and the path length is ℓ, then the message complexity is ℓn^2. Typically, for a DHT of η nodes, $n = \Theta(\log \eta)$ and, as in Chord [SMK+01], $\ell = O(\log \eta)$ which gives $O(\log^3 \eta)$ messages; this is likely too costly for practical values of η.

Saia and Young [SY08] give a randomized protocol which provably achieves $O(\log^2 \eta)$ messages in expectation. While communication between two quorums incurs an expected constant number of messages, the analysis in [SY08] yields a prohibitively large constant. The protocol also employs a link architecture between peers requiring the use of a Byzantine agreement protocol. Finally, maintenance and asynchronicity issues remain unresolved.

Therefore, while results exist on the feasibility of robust communication, work on the practicalities has lagged behind. This dearth presents an impediment to the deployment of such systems. In this chapter, we discuss our recent results [YKGK10] that address this outstanding problem.

Contributions. In the computational setting, for an adversary that controls up to an $\rho < 1/3$-fraction of any quorum of size at most n, we present two efficient protocols for achieving robust communication of a message m to a set of peers $D \subseteq \mathsf{Q}_i$ for some quorum Q_i over a path of length ℓ.

For our Robust Communication Protocol I (RCP-I), the total message complexity and the message complexity of the sending peer is each at most $2n + 4n(\ell - 2) + |D|$. The message complexity of every non-sending peer along the lookup path is at most 4 and the communication latency is at most $2(\ell - 2) + 2$. For our Robust Communication Protocol II (RCP-II), the expected total message complexity and the expected message complexity of the sending peer is each at most $2s + \frac{(\ell-2)}{(1-\rho)c} + (\ell - 2) + |D|$. The expected message complexity of a non-sending peer on the lookup path is at most $\frac{2}{(1-\rho)cn}$ and the expected latency is at most $\frac{(\ell-2)}{(1-\rho)c} + 2$. Here, the constant $c > 0$ is the probability that the response time of an honest peer is at most some constant value Δ.

Using the Chord-based construction of [FSY05], the message complexity of RCP-I is $O(\log^2 \eta)$ and for RCP-II it is $O(\log \eta)$ in expectation. We tolerate a large fraction of adversarial peers; strictly less than a 1/3-fraction compared to the roughly 1/4-fraction in [SY08]. Our use of a distributed key generation (DKG) scheme allows for security *without* a trusted party or costly updating of public/private keys outside of each quorum. To the best of our knowledge, this is the first use of DKG in a Byzantine-tolerant P2P setting.

Finally, we provide microbenchmark results involving two quorums using PlanetLab. Our experimentation demonstrates that our protocols perform well under significant levels of churn and faulty behaviour. In particular, for a 10^5-node system with $\ell = 20$, our results imply RCP-I and RCP-II complete in under 4 seconds and 5 seconds, respectively.

Related Work. State machine replication is a standard method for implementing highly fault-tolerant services [Sch90]. Services are replicated and run over multiple servers providing a high-integrity distributed system whereby operations can be invoked by clients. Over the past several years, a large body of literature has been established on implementing Byzantine fault-tolerant replication protocols. Peer-to-peer systems do not align perfectly with the state machine replication paradigm; however, these results are relevant in the context of implemented Byzantine fault-tolerant systems.

Pioneering work by Reiter [Rei95] yielded a toolkit of protocols for Byzantine agreement and atomic broadcast. More recently, Castro and Liskov [CL01, CL02] demonstrated that Byzantine fault tolerant state machine replication could be accomplished while maintaining satisfactory system performance. However, their protocol is unsuitable for peer-to-peer environments due to issues of scaling. Several other Byzantine fault-tolerant systems have been implemented such as SINTRA [CP02], FARSITE [ABC$^+$02], the Query/Update (Q/U) protocol [AEMGG$^+$05] and the HQ system [CML$^+$99]; however, scalability is an obstacle to the adaptation of these protocols to peer-to-peer environments.

Two implemented large-scale Byzantine fault tolerant storage architectures are the Oceanstore [KBC$^+$00] and Rosebud [RL03]. The latter system scales for up to tens of thousands of nodes and allows for changing membership. However, experimental results indicate that Rosebud may tolerate only low numbers of Byzantine faults in practice. Another concern is that the system relies on a crucial component known as the *configuration service* (CS) which is responsible for tracking system membership, ejecting faulty nodes, and handling new nodes; a similar component known as a *primary tier* of replicas is used in Oceanstore. The CS can be implemented over a set of nodes; however, the CS introduces a hierarchy that can become a bottleneck for the system and a point of vulnerability.

There are proposals for applying the state machine replication approach on a large scale. In [RKB07], the authors propose the ShowByz system which utilizes a set of nodes acting as a *configuration manager* whose job is to track system members, addresses, public keys and the responsibilities of replica groups. A similar concept is proposed by Rodrigues *et al.* in [RLS02]; a peer-to-peer system is proposed that relies, again, on a configuration service. Neither work provides empirical results and the details of secure data retrieval and message passing are not discussed.

There are a number of theoretical results on Byzantine fault-tolerance. As previously mentioned, there is a large body of literature describing DHTs that can tolerate adversarial faults [FS02, NW03, HK04, FSY05, AS06b]. These results make use of *quorums*, which are sets of $\Theta(\log \eta)$ peers with a majority of the peers in a quorum being honest. However, an adaptive adversary may have its peers join and leave the network until obtain a some quorum has a majority of faulty peers. The current state of the art in protecting against such attacks is due to results by Awerbuch and Scheideler [AS06a, AS06b, AS07] which describe a DHT that remains robust even if the number of join and leave events is polynomial in the size of the network. Recent work by Saia and Young [SY08] shows how routing with quorums can be made less resource intensive; however, as discussed earlier, several issues remain unresolved as obstacles to deployment.

Castro *et al.* [CDG+02], Halo [KT08], and Salsa [NW06] handle Byzantine faults by routing along multiple diverse routes. The proposal in [CDG+02] requires a CA whereas we do not rely on any trusted third party. In both [KT08] and [NW06], the guarantees are unclear against an adversary who owns a large IP-address space or adaptively targets identifiers over time as described in [AS06b]. In contrast, defences for quorum-based protocols are known [AS06a, AS06b, AS07].

9.2 Quorum Topology and Threshold Cryptography

Each peer P is assumed to have a unique identifier, $P_{\mathtt{ID}}$, and a network address, P_{addr}. Peers P and Q are said to communicate directly if each has the other in its routing table. The target of m is a set of peers D within a single quorum; m may be a data item request and D may consist of a single peer or multiple peers depending on how data is stored.

Quorum Topology. There are several different approaches to how quorums are created and maintained [NW03, AS06b, SY08]. Despite these different approaches, we may view the setup of quorums as a graph where nodes correspond to quorums and edges correspond to communication capability between quorums; we refer to this as the *quorum topology*. We assume the following four simple invariants are true:

Goodness. each quorum has size $n = \Omega(\log \eta)$ and possesses at most an ρ-fraction of Byzantine peers for $\rho < 1/3$.

Membership. every peer belongs to at least one quorum.

Intra-Quorum Communication. every peer can communicate directly to all other members of its quorums.

Inter-Quorum Communication. if Q_i and Q_j share an edge in the quorum topology, then $P \in \mathsf{Q}_i$ may communicate directly with any member of Q_j and *vice versa*.

These four invariants are standard in the sense that previous works on quorums in DHTs ensure they hold with high probability. For example, results for maintaining the goodness invariant in DHTs are known [AS06a, AS06b, AS07]. In terms of the membership invariant, there exist quorum topologies where a peer may belong to several different quorums simultaneously [NW03, FSY05].

Finally, to the best of our knowledge, no implementation of a quorum topology exists; this represents another gap between theory and practice. A number of challenges remain in bridging this gap and such an endeavour is outside the scope of this current work. However, the literature suggests that, with the proper deployment, maintaining these four invariants in real-world DHTs is plausible.

Assumptions. We adopt our Hybrid system model from Section 4.2 and work in the asynchronous communication model with a Byzantine adversary, and crashes and link failures. For liveness in HybridDKG and in the RCP-II protocol, we use the weak synchrony assumption by Castro and Liskov [CL02].

The adversary is assumed to have full knowledge of the network topology and control all faulty peers, which forms a constant fraction of all nodes in the system. In concert with the goodness invariant, strictly less than 1/3 of the peers in any quorum can be faulty. These peers may collude and coordinate their attacks. Our adversary is computationally bounded with the security parameter κ and it has do 2^κ computation to solve the Gap Diffie-Hellman (GDH) problem [BLS01] in an appropriate group.

Threshold Signatures. We use threshold signatures to authenticate the communication between quorums. In an (n, t)-threshold signature scheme, a signing (private) key k is distributed among n parties by a trusted dealer using VSS or in a dealerless fashion using DKG. Along with private key shares k_i for each party, the distribution algorithm also generates a verification (public) key K and the associated public key shares $\widehat{K}$. To sign a message m, any subset of $t + 1$ or more parties use their shares to generate the signature shares σ_i. Any party can combine these signature shares to form a message-signature pair $S = (m, \sigma) = [m]_k$ that can be verified using the public key K; however, this does not reveal k. We refer to a message-signature pair S as a signature. It is also possible to verify the individual signature shares σ_i using the public key shares $\widehat{K}$. We assume that no computationally bounded adversary that corrupts up to t parties can forge a signature $S' = (m', \sigma')$ for a message m'. Further, malicious behaviour by up to t parties cannot prevent generation of a signature.

Many threshold signature schemes have been constructed in the literature. However, the threshold signature schemes other than [GJKR96, Sho00, Bol03] are not useful for a variety of reasons such as impractical adversary assumptions. Out of these three practical schemes, the threshold DSS scheme [GJKR96] requires a significant amount of interaction among all the parties for every signature generated, while removing the requirement of a trusted dealer is a difficult multiparty computation problem in the threshold RSA signature scheme [Sho00]. The threshold version [Bol03] of the Boneh-Lynn-Shacham (BLS) signature scheme [BLS01] avoids all of the above problems. Its key generation does not mandate a trusted dealer. The signature generation protocol does not require any interaction among the signing parties or any zero-knowledge proofs. Further, the BLS signature size and generation algorithm are more efficient than RSA and DSS signatures. Therefore, to authenticate the communication between the quorums, we use the threshold BLS signature scheme. This scheme uses the bilinear pairings setting and its security is based on the difficulty of solving the GDH problem (refer to [Bol03] for a detailed description).

In absence of a trusted party in the P2P paradigm, we use our DKG scheme to generate the (distributed) private key for each quorum. Our HybridDKG protocol is the first DKG for an asynchronous setting; therefore, it is uniquely suitable for a P2P network. Along with a Byzantine adversary, this protocol also tolerates crash failures. For a quorum of size n, with t Byzantine nodes and f honest nodes that can crash, the DKG protocol requires that $n \geq 3t+2f+1$. In our case, this *security threshold* holds due to the goodness invariant.

9.3 Our Robust Communication Protocols

We propose two robust communication protocols: RCP-I and RCP-II. Here we outline a general scheme in Figure 9.1 that is later refined to give our two protocols. Consider a sending peer P who wishes to send a message m to peer Q. We assume m is associated with a key value which yields information necessary for distributed routing; that is, the next peer to which m should be forwarded is always known. Peer P notifies its quorum Q_1 that it is performing robust communication and receives $\textsc{Proof}(\mathsf{Q}_1)$ that shows that P's actions are authorised by quorum Q_1. Peer P sends this to Q_2 as proof that P's actions are legitimate; the form of this proof is discussed later. Depending on the scheme, one or more members of Q_2 examines the proof and, upon verifying it, sends to P: (1) routing information for Q_3 and (2) $\textsc{Proof}(\mathsf{Q}_2)$, which will convince Q_3 that P's actions are legitimate. This continues iteratively until P contacts the quorum holding Q and m is delivered. We employ the following concepts:

Quorum Public/Private Keys: Each quorum Q_i is associated with a (distributed) public/private key pair $(K_{\mathsf{Q}_i}, k_{\mathsf{Q}_i})$; however, there are two crucial differences between how such a key pair is utilized here in comparison to traditional architectures. First, only those quorums linked to Q_i in the quorum topology, and not everyone in the network, need to know K_{Q_i}. Second, $(K_{\mathsf{Q}_i}, k_{\mathsf{Q}_i})$ is created using the DKG protocol and $\widehat{K}_{\mathsf{Q}_i}$ is the associated set of public key shares.

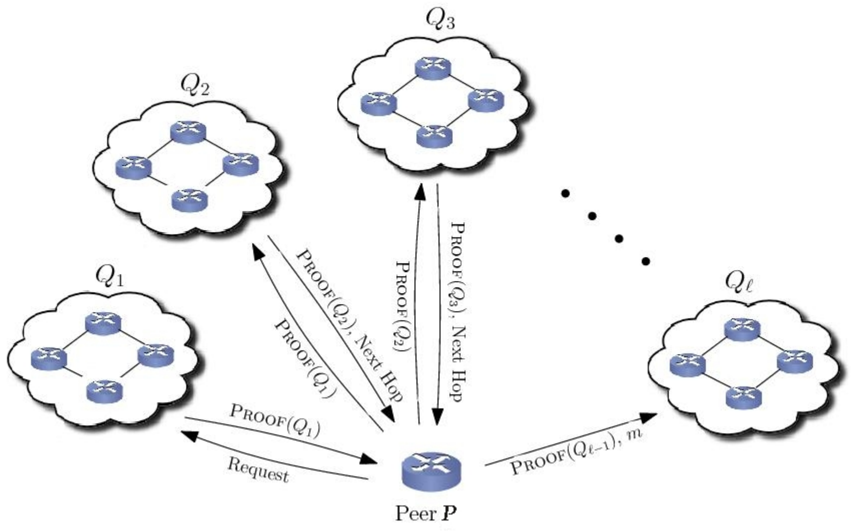

Figure 9.1: Our general robust communication scheme. At step $i = 1, \ldots, \ell - 1$, peer P presents proof, Proof(Q_i), that quorum Q_i sanctions P's action, and receives new proof from Q_{i+1} in addition to routing information for the next hop. At the final step ℓ,peer P sends Proof$(\mathsf{Q}_{\ell-1})$ and m.

Individual Public/Private Key Shares: Each peer $P \in \mathsf{Q}_i$ possesses a private key share $(k_{\mathsf{Q}_i})_P$ of k_{Q_i} produced using DKG. Unlike the quorum public/private key pair of Q_i which must be known to all quorums to which Q_i is linked in the quorum topology, only the members of Q_i need to know the corresponding public key shares $\widehat{K}_{\mathsf{Q}_i}$, which allow members of Q_i to verify the validity of the signature shares sent to P.

9.3.1 Robust Communication Protocol I

In this section, we outline RCP-I. The path m takes through quorums is denoted by $\mathsf{Q}_1, \ldots, \mathsf{Q}_\ell$. We assume that $P \in \mathsf{Q}_1$ and the target of the message is a set of peers $D \subseteq \mathsf{Q}_\ell$. Initially, the honest peers of Q_1 must acquiesce to P's request. Peer P begins by sending $[P_{\text{ID}}, P_{\text{addr}}, \texttt{key}, ts_1]$ to all peers in its quorum Q_1. The value key corresponds to the intended destination of m and ts_1 is a time stamp. The message m can also be sent, and its hash can be added inside the signature below; however, for simplicity, we assume m is sent only in the last step. Each honest peer $Q \in \mathsf{Q}_1$ then sends its signature share to P if P is not in any quorum-level violation. Peer P interpolates these signature shares to generate the signature: $S_1 \leftarrow [P_{\text{ID}}, P_{\text{addr}}, \texttt{key}, ts_1]_{k_{\mathsf{Q}_1}}$.

In each intermediate step $i = 2, \ldots, \ell - 1$, P sends its most recent signature S_{i-1} and a new time stamp ts_i to each peer $Q \in \mathsf{Q}_i$ along the lookup path. Since Q_i is linked to Q_{i-1} in the quorum topology, each Q knows the public key $K_{\mathsf{Q}_{i-1}}$ to verify S_{i-1}. If S_{i-1} is verified and ts_i is valid, Q sends back its signature share, $K_{\mathsf{Q}_{i+1}}$ and the routing information. Peer P collects the shares to form S_i and majority filters on the routing information for Q_{i+1}. Finally, for Q_ℓ, P sends m along with $\mathcal{S}_{\ell-1}$ to peers in the target set D. More details of the protocol are provided in [YKGK10].

9.3.2 Robust Communication Protocol II

RCP-II utilizes signed routing table information. As a concrete example, we assume a Chord-like DHT although other DHT designs can be accommodated. For a peer $U \in \mathsf{Q}_i$, each entry of its routing table has the form $\mathcal{RT}_{\mathsf{Q}_j} = [\mathsf{Q}_j, P_{\mathtt{ID}}, P'_{\mathtt{ID}}, K_{\mathsf{Q}_j}, ts]$. Here $P \in \mathsf{Q}_j$ and $P' \in \mathsf{Q}_{j-1}$ where (1) Q_i links to Q_j and Q_{j-1} in the quorum topology, (2) Q_{j-1} immediately precedes Q_j clockwise in the identifier space and (3) P and P' are respectively located clockwise of all other peers in Q_j and Q_{j-1}. K_{Q_j} is the quorum public key of Q_j, and ts is a time stamp for when this entry was created. Note that any point in the identifier space falls between unique points P_{ID} and P'_{ID}. Given this property, and that entries are signed by a quorum, any attempt by a malicious peer along the lookup path to return incorrect routing information can be detected. $[K_{\mathsf{Q}j}]_{k_{\mathsf{Q}_i}}$ is the quorum public key of Q_j signed using the private quorum key of Q_i; recall that neighbours in the quorum topology know each others' public keys. $[\mathcal{RT}_{\mathsf{Q}_j}]_{k_{\mathsf{Q}_i}}$ is the routing entry for Q_j signed with the private key of Q_i; entries of the routing table are signed separately. Routing table information is time stamped and re-signed periodically when the HybridDKG share renewal protocol (see Section 5.2.1) is executed.

We next outline RCP-II. For simplicity, we temporarily assume that peers act correctly; see [YKGK10] for a full specification and discussion of the protocol. Initially, each correct peer in Q_1 receives $[P_{\mathtt{ID}}, P_{\mathrm{addr}}, \mathtt{key}, ts]$ from P. The time stamp ts is chosen by P and peers in Q_1 will acquiesce to the value if it agrees with the rule set to within some bound to compensate for clock drift. If the request does not violate the rule set, then the information is signed allowing P to form $M_1 = [P_{\mathtt{ID}}, P_{\mathrm{addr}}, \mathtt{key}, ts]_{k_{\mathsf{Q}_1}}$.

In the second step of the protocol, P knows the membership of Q_2 and selects a peer $Q_2 \in \mathsf{Q}_2$ uniformly at random without replacement. Peer P then sends M_1 to Q_2. Assuming Q_2 is correct, it verifies M_1 using K_{Q_1} and checks that the ts is valid. Once verified, Q_2 sends P the information $[K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}$, $[\mathcal{RT}_{\mathsf{Q}3}]_{k_{\mathsf{Q}_2}}$ and $[K_{\mathsf{Q}3}]_{k_{\mathsf{Q}_2}}$. Peer P knows K_{Q_2} since Q_1 links to Q_2 and verifies $[K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}$, $[\mathcal{RT}_{\mathsf{Q}3}]_{k_{\mathsf{Q}_2}}$ and $[K_{\mathsf{Q}3}]_{k_{\mathsf{Q}_2}}$, and checks that the time stamp on the routing information is valid. If so, P constructs $M_2 = [M_1, [K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}]$. Here $[K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}$ will allow some peer in Q_3 to verify K_{Q_1} and M_1, while the signed verified $K_{\mathsf{Q}3}$ will allow P to check the response from that peer in Q_3.

This process repeats with minor changes for the remaining steps. Using $\mathcal{RT}_{\mathsf{Q}_3}$ from the previous step, P selects a peer Q_3 randomly from Q_3 and sends M_2. Since Q_3 is linked with Q_2 in the quorum topology, Q_3 knows $K_{\mathsf{Q}2}$, which it uses to verify $[K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}$; this allows Q_3 to verify M_1 signed with $k_{\mathsf{Q}1}$. Peer Q_3 then confirms that ts is valid and sends $[K_{\mathsf{Q}2}]_{k_{\mathsf{Q}_3}}$, $[\mathcal{RT}_{\mathsf{Q}_4}]_{k_{\mathsf{Q}_3}}$ and $[K_{\mathsf{Q}4}]_{k_{\mathsf{Q}_3}}$ to P. Peer P has a verified public key $K_{\mathsf{Q}3}$ from the previous step and uses it to verify $[K_{\mathsf{Q}2}]_{k_{\mathsf{Q}_3}}$, $[\mathcal{RT}_{\mathsf{Q}_4}]_{k_{\mathsf{Q}_3}}$, and $[K_{\mathsf{Q}4}]_{k_{\mathsf{Q}_3}}$. Then P constructs $M_3 = [M_2, [K_{\mathsf{Q}2}]_{k_{\mathsf{Q}_3}}] = [M_1, [K_{\mathsf{Q}1}]_{k_{\mathsf{Q}_2}}, [K_{\mathsf{Q}2}]_{k_{\mathsf{Q}_3}}]$ and sends that to a peer Q_4 selected randomly from Q_4. This process continues until P reaches the destination quorum Q_ℓ. For Q_ℓ, P sends m along with $M_{\ell-1}$ to peers in the target set D. The full details of the protocol are provided in [YKGK10].

9.4 Experimental Results

In this section, we examine the performance of our two protocols over PlanetLab. Based on our experimental results and known churn rates, we propose parameters for DHTs using our protocols.

9.4.1 Microbenchmarks

Distributed Key Generation. DKG is a crucial component of our protocols. It is required to initiate a threshold signature system in a quorum and to securely manage membership changes. We use our HybridDKG implementation from Chapter 8. We incorporate threshold BLS signatures into this implementation and realize our two protocols using this setup on PlanetLab. We use the completion time and median CPU usage values from Table 8.1 for quorum sizes $n = 10, 15, 20, 25, 30$. The median completion periods vary from 6 seconds for $n = 10$ to more than 5 minutes for $n = 30$. As we observe in Section 8.2, the bulk of this latency is due to network delays; in contrast, the required CPU time is far smaller than the completion periods.

In the next subsection, we examine the feasibility of these completion periods. Our DKG experiments assume that 30% of the peers may crash and 10% of the peers may be Byzantine. While we can tolerate any fraction of Byzantine peers less than 1/3, we use these numbers since in many practical scenarios we expect the fraction of Byzantine faults to be less than 10% and modest compared to the fraction of crash failures.

RCP-I and RCP-II. For our RCP-I and RCP-II experiments, we set $n = 30$, $t = 3$, and $f = 10$. In RCP-I, a node requires 0.14 seconds on average to obtain a threshold signature from a quorum, if all of the obtained signature shares are correct. The average execution time increases to 0.23 seconds in case of a share corruption attack. Extrapolating to a path length ℓ, an operation should take 0.14ℓ to 0.23ℓ seconds on average. With 10^5 nodes, the average total time for RCP-I is then 3 to 4 seconds with $\ell = 20$.

In RCP-II, a node takes 0.04 seconds on average to obtain the required signed public keys and the signed routing information from a correct peer. A single signature verification takes 0.004 seconds on average. The median latency value over PlanetLab is roughly 0.08 seconds [DLS+04]; half of the total number of messages over PlanetLab get delivered in 0.08 seconds and the message transmission delay $\Delta = 0.08$ seconds for probability $c = 0.5$. With a chain of signed public keys of length ℓ, the total communication time is $0.14 + 0.04(\ell - 1) + \frac{\Delta(\ell-2)}{c(1-\rho)} + 0.004\frac{\ell(\ell-1)}{2}$ which for 10% Byzantine peers, is 4.68 seconds in expectation. To a first approximation, the execution times of our protocols seem quite reasonable.

System Load. We address the issue of system load under the assumption that signature verification is the most significant computational operation. We make back-of-the-envelope calculations to obtain the expected order of magnitude for our performance figures. For RCP-I, from the above discussion, each signature verification takes 0.004 seconds; thus, the total CPU time required per execution is $0.004\ell(1 + n + n^2)$; this includes the costs due to share corruption attacks. For $\ell = 20$ and $n = 30$, this value is 75 CPU seconds, spread out over 600 nodes. Therefore, the number of executions of RCP-I that can be started per second on average is $n/75 \approx 10^3$ when $n = 10^5$. Note this rate value is for *the entire system*. Now, if no share corruption attacks occur, the total CPU time required per execution becomes $0.004\ell(1+n)$ which, for the same parameter values, is 2.5 CPU seconds. This implies that $4 \cdot 10^4$ executions can be started per second on average in the entire system. For RCP-II, the total CPU time required for execution is given by $0.004\left(\ell + \frac{(\ell-1)\ell}{2(1-\rho)}\right)$ which, for the same parameters and $\rho = 1/10$ is roughly 1 CPU second on average. Therefore, approximately 10^5 executions can be started per second on average in the entire system.

9.4.2 Analysis and Discussion

As mentioned in Section 9.2, important questions remain with regards to translating theoretical results to a practical setting. In particular, two quantities of interest are the size of quorums, n, and the number of quorums to which each peer belongs, n_{Q}. Unfortunately, pinning down these quantities is non-trivial; only asymptotic analysis is present in the literature. Furthermore, it is not a simple case of substituting hard numbers because n depends on a number of parameters: (1) the exact guarantees being made, (2) algorithms for quorum maintenance, (3) the tools of analysis (i.e. form of Chernoff bounds used) and many more. Evaluating these parameters is outside the scope of this work. Instead, we

Table 9.1: The expected number of seconds before a quorum experiences a membership change (r_{Q})

n	10			15			20			25			30		
n_{Q}	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
r_{Q}	526	351	175	350	234	117	263	132	88	210	140	70	175	87	58

assume a range of values for n and n_{Q}. As our protocols appear to be the most efficient to date, the following results illuminate what currently seems possible in practice.

System Churn and DKG. The performance of our two protocols will likely depend on system churn. A common metric for measuring the degree of churn is *session time*: the time between when a node joins the network and when it leaves [RGRK04]. In this work, we make the standard assumption that the cost of joining the network is large enough so as to prevent the adversary from substantially increasing the rate of churn through rapid rejoin operations. This can be achieved using monetary costs as in [CDG+02] or CAPTCHAs as suggested in [NW06].

Argument for Batching. Investigations have yielded differing measurements for median session times. The Kazaa system was found to have a median session time of 144 seconds [GDS+03]. In the Gnutella and Napster networks, the median session time was measured to be approximately 60 minutes [SGG02]. Measurements of the Skype P2P network yielded a median session time of 5.5 hours for super-peers [GDJ06]. Here, we temporarily assume a median session time of 60 minutes and a standard Poisson model of peer arrivals/departures as in [LNBK02, RGRK04]. To calculate churn rate, r (number of arrivals/departures per second), based on the median session time t_{med} (in seconds), we use the formula of [RGRK04]: $r = (\eta \cdot \ln 2)/t_{med}$. For $\eta = 10^5$ and $t_{med} = 3600$ seconds, $r \approx 19$. Assuming that join and leave events occur independently of each other, Table 9.1 gives the expected number of seconds, r_{Q}, at which point a quorum will undergo a membership change when each peer belongs to n_{Q} quorums. Our choice of $n_{\mathsf{Q}} \leq 3$ is based upon the reasonable assumption that overlap occurs only with immediate neighbouring quorums in the ID space.

In several cases, the r_{Q} values are less than the corresponding median DKG completion times in Table 8.1. Therefore, a quorum may not be able to execute DKG often enough to accommodate each membership change. However, join operations can be queued and performed in batches. Executing DKG for a batch of joins does not increase the message complexity and message size increases only linearly in the batch size (see Chapter 5). Therefore, batching can mitigate the effects of churn and it seems plausible that peers

Table 9.2: Median session times (in seconds) derived from values for n, n_{Q} and r_{DKG} (in seconds)

n	10			15			20			25			30		
r_{DKG}	600			900			1200			1500			1800		
n_{Q}	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
t_{med}	1400	2800	4199	2386	4772	7159	2930	5859	8789	4433	8867	13300	4442	8883	13325

would tolerate some delay in joining in exchange for security.

Batching and the Security Threshold: Batching join events improves performance; however, many peers might leave a quorum before a new batch is added, thus violating the security threshold. Hence, we are interested in the session time value required such that this is not likely to occur. Based on Table 8.1 for $n = 20$ and $n_{\mathsf{Q}} = 1$, DKGs complete within 68 seconds. The number of leave events a quorum can suffer while not exceeding the crash limit is $f = 6$. If Byzantine peers leave, more crashes are tolerable; however, identifying such events is impossible, so we assume the worst case of $f = 6$. Assuming DKG executes every $r_{DKG} = 1200$ seconds, we seek the median session time such that at most 6 peers leave the system within 1268 seconds. With $\eta/n = 5000$ quorums in the system, each experiencing 6 leave events within 1268 seconds, the system churn rate is $r = 6 \cdot 5000/1268 = 23.7$. This gives $t_{med} = (\eta \cdot \ln 2/r) = 2930$ or, equivalently, 49 minutes. Therefore, with this t_{med}, we expect the system to remain secure. Moreover, a quorum only spends $68/1268 = 5.4\%$ of the time executing DKG.

Certain parameters can be tuned to offer performance trade-offs. Decreasing r_{DKG} yields smaller required median session times; however, the percentage of time spent on DKG increases. Such tuning would depend on the desired system performance, the application, and n and n_{Q}. Table 9.2 gives session time calculations for other values of n, r_{DKG} and n_{Q}. Our choice of $n_Q \leq 3$ is based upon the assumption that quorums only overlap with their immediate neighbours in the ID space.

Required session times increase with n. Notably, for $n = 30$ and $n_{\mathsf{Q}} = 1$, t_{med} does not far exceed the 60 minutes in [SGG02]. As n_{Q} increases, the required session times grow linearly. However, our maximum of 3.7 hours is still less than the t_{med} measured for super-peers in the Skype network [GDJ06]. We tentatively conclude that our protocols can be deployed in applications where session times range from 10 minutes to a few hours and that such applications currently exist.

Finally, the implementation of the DKG protocol used here is an academic version, and more efficient implementations may be possible. In terms of performance improvements, using the aggregate signature scheme [BGLS03], messages can be made more compact yielding a savings in RCP-II. For certain applications, it may be possible to restrict membership to those peers that meet certain latency and bandwidth criteria. In terms of fault-tolerance, our experiments were performed with 10% of nodes suffering Byzantine faults; however, we generally expect the fraction of Byzantine nodes to be less, thus reducing execution times.

The performance of a complete system is an important open question. The quorum topology chosen is crucial and optimizing this in practice is a topic of future work. While we focus on DHTs, our results likely apply to other P2P designs and more general settings where groups of machines, some with untrustworthy members, must communicate; it would be of interest to identify more such applications.

Chapter 10

Conclusions

The most important conclusion that we draw from this thesis is that the practical distributed key generation is possible for use in the Internet-scale applications.

Distributed Key Generation. 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 the existing Internet. We also incorporated crash-recoveries and network failures in the system along with the traditional Byzantine adversary.

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

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 long-term system sustainability and proposed protocols to achieve group modification agreement, node addition, node removal, and security-threshold and crash-limit modification.

We also made a cryptographic contribution by defining a constant-size commitment scheme (PolyCommit) for polynomials and extended it to define a synchronous VSS scheme that requires constant-size broadcast instead of the linear broadcast required previously. We then extended it to define an efficient synchronous DKG scheme. We are currently working towards using a similar commitment technique in the asynchronous communication model to reduce the bit complexity of our HybridVSS protocol.

In future, we plan to make our HybridDKG protocol secure against adaptive adversaries by introducing the rewinding adversary similar to the one in [CGJ+99]. Further, we use the random oracle assumption to achieve uniform randomness of the shared secret in HybridDKG. It would be interesting to obtain uniform randomness without random oracles using some distributed commitment techniques or the common reference string model.

Applications. In this thesis, we worked on two cryptographic applications of HybridDKG and one system-level application of our HybridDKG implementation.

As the first application, we designed and compared distributed PKG setup and private key extraction protocols for Boneh and Franklin's BF-IBE, Sakai and Kasahara's SK-IBE, and Boneh and Boyen's BB_1-IBE. Each of the above three schemes represents a separate category of IBE schemes and our designs can be applied to other schemes in those categories as well. In terms of practical use, we observed that the distributed PKG implementation for BF-IBE is the most simple and efficient among all and we suggest its use when the system can support its relatively costly encryption step. For systems requiring a faster encryption, we suggest the use of BB_1-IBE instead. However, during every distributed private key extraction, it requires a DKG and consequently, interaction among PKG nodes. That being said, during private-key extractions, we successfully avoid any interaction between clients and PKG nodes except the necessary identity at the start and key share transfers at the end.

As the second application, we presented new identity-based approaches for circuit construction in onion routing anonymity networks. We defined one-way and two-way anonymous and pseudonymous key agreement protocols in the BF-IBE setting and used this scheme to produce new onion routing circuit construction protocols. Our distributed PKG for BF-IBE solves the key escrow problem in this setting. Our single pass circuit construction uses significantly less computation and communication than the corresponding protocol in Tor, and reduces the load on the network support infrastructure. To achieve immediate forward secrecy instead of eventual forward secrecy, we also defined λ-pass circuit construction. These improvements can be used to improve the efficiency and to enhance the scalability of low-latency anonymity networks.

Observing that the Sphinx message format defined for mix networks is also applicable to OR circuit constructions, we designed a generic OR circuit construction that is compact as well as secure in the UC model. Further, we used this generic construction to improve the circuit constructions for the PB-OR protocol. From a practical perspective, we then compared the messages in the new circuit constructions with the original protocol and noted that the new messages are significantly smaller.

Finally, we used our HybridDKG implementation to define two robust communication protocols for DHTs that do not require any trusted third party. These protocols, using the threshold BLS signature scheme, asymptotically improve the communication cost of robust communication in the quorum-based DHTs with a Byzantine adversary. Our first protocol is deterministic and achieves $O(\log^2 \eta)$ message complexity and our second protocol is randomized and achieves $O(\log \eta)$ message complexity in expectation for DHTs of size η. We performed experiments over PlanetLab involving two quorums and provided the microbenchmark results. Our experimentation demonstrates that our protocols perform well under significant levels of churn and faulty behaviour.

References

[ABC$^+$02] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted environment. In *OSDI'02*, pages 1–14, 2002. 141

[ACF09] M. Abdalla, D. Catalano, and D. Fiore. Verifiable Random Functions from Identity-Based Key Encapsulation. In *Advances in Cryptology—EUROCRYPT'09*, pages 554–571, 2009. 76

[ADH08] I. Abraham, D. Dolev, and J. Y. Halpern. An Almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In *ACM PODC'08*, pages 405–414, 2008. 40

[ADW09] J. Alwen, Y. Dodis, and D. Wichs. Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. In *Advances in Cryptology—CRYPTO*, pages 36–54, 2009. 31

[AEMGG$^+$05] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-Scalable Byzantine Fault-Tolerant Services. In *SOSP'05*, pages 59–74, 2005. 141

[AF04] M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In *Advances in Cryptology—CRYPTO'04*, pages 317–334, 2004. 11

[ARP03] S. S. Al-Riyami and K. G. Paterson. Certificateless Public Key Cryptography. In *Advances in Cryptology—ASIACRYPT'03*, pages 452–473, 2003. 77

[ARS07] J. Aspnes, N. Rustagi, and J. Saia. Worm versus alert: Who Wins in a Battle for Control of a Large-Scale Network? In *OPODIS'07*, pages 443–456, 2007. 139

[AS06a] B. Awerbuch and C. Scheideler. Robust Random Number Generation for Peer-to-Peer Systems. In *OPODIS'06*, pages 275–289, 2006. 139, 142, 143

[AS06b] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *SPAA'06*, pages 318–327, 2006. 139, 142, 143

[AS07] B. Awerbuch and C. Scheideler. Towards Scalable and Robust Overlay Networks. In *IPTPS'07*, 2007. 139, 142, 143

[ASM08] M.H. Au, W. Susilo, and Y. Mu. Practical anonymous divisible e-cash from bounded accumulators. In *FC'08*, pages 287–301, 2008. 25, 26

[AWL05] M. Afergan, J. Wein, and A. LaMeyer. Experience with Some Principles for Building an Internet-Scale Reliable System. In *WORLDS'05*, pages 1–6, 2005. 39

[AWSM07] M. H. Au, Q. Wu, W. Susilo, and Y. Mu. Compact E-Cash from Bounded Accumulator. In *CT-RSA'07*, pages 178–195, 2007. 25, 26

[BB04a] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In *Advances in Cryptology—EUROCRYPT'04*, pages 223–238, 2004. 15

[BB04b] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology—EUROCRYPT'04*, pages 56–73, 2004. 24, 25, 26, 93, 99

[BBG05] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology—EUROCRYPT'05*, pages 440–456, 2005. 15, 25

[BC03] M. Backes and C. Cachin. Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries. In *IEEE DSN'03*, pages 37–46, 2003. 43, 44, 45, 48, 56, 68

[BCS03] M. Backes, C. Cachin, and R. Strobl. Proactive Secure Message Transmission in Asynchronous Networks. In *ACM PODC'03*, pages 223–232, 2003. 65

[BdM93] J.C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In *Advances in Cryptology—EUROCRYPT'93*, pages 274–285, 1993. 25

[BF01] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology—CRYPTO'01*, pages 213–229, 2001. 2, 5, 39, 75, 76, 78, 83, 85, 86, 87, 89

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *ACM STOC'88*, pages 103–112, 1988. 36

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology—EUROCRYPT'03*, pages 416–432, 2003. 151

[BIB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *ACM PODC'89*, pages 201–209, 1989. 79, 81

[Bla79] G. R. Blakley. Safeguarding Cryptographic Keys. In *the National Computer Conference*, pages 313–317, 1979. 1, 10

[BLS01] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Advances in Cryptology—ASIACRYPT'01*, pages 514–532, 2001. 143, 144

[BM07] X. Boyen and L. Martin. Identity-Based Cryptography Standard (IBCS) (Version 1), Request for Comments (RFC) 5091. `http://www.ietf.org/rfc/rfc5091.txt`, 2007. 5, 78, 83, 93, 95

[BOCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *ACM STOC'93*, pages 52–61, 1993. 51

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (ext. abstract). In *ACM STOC'88*, pages 1–10, 1988. 34, 79

[Bol03] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC'03*, pages 31–46, 2003. 77, 143, 144

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *ANTS-III*, pages 48–63, 1998. 15

[Boy07] X. Boyen. General *Ad Hoc* Encryption from Exponent Inversion IBE. In *Advances in Cryptology—EUROCRYPT'07*, pages 394–411, 2007. 83

[Boy08] X. Boyen. A Tapestry of Identity-based Encryption: Practical Frameworks Compared. *Int. J. Appl. Cryptol.*, 1(1):3–21, 2008. 5, 77, 78, 83, 93, 95, 98

[Bra84] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *ACM PODC'84*, pages 154–162, 1984. 41

[BSS05] I. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography.* Number 317 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2005. 183–252. 16

[Can01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE FOCS'01*, pages 136–145, 2001. 6, 102

[CC05] L. Chen and Z. Cheng. Security Proof of Sakai-Kasahara's Identity-Based Encryption Scheme. In *IMA Int. Conf. on Cryptography and Coding*, pages 442–459, 2005. 90, 91, 92

[CDG$^+$02] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *OSDI'02*, pages 299–314, 2002. 142, 149

[CDvdG87] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result. In *Advances in Cryptology—CRYPTO'87*, pages 87–119. Springer, 1987. 17

[CFG09] D. Catalano, D Fiore, and R. Gennaro. Certificateless Onion Routing. In *ACM CCS'09*, pages 151–160, 2009. 121

[CFM08] D. Catalano, D. Fiore, and M. Messina. Zero-Knowledge Sets With Short Proofs. In *Advances in Cryptology—EUROCRYPT'08*, volume 4965, pages 433–450, 2008. 24

[CGJ$^+$99] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *Advances in Cryptology—CRYPTO'99*, pages 98–115, 1999. 11, 14, 18, 154

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *IEEE FOCS'85*, pages 383–395, 1985. 3, 10

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Advances in Cryptology—EUROCRYPT'97*, pages 103–118, 1997. 39

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 4(2):84–88, 1981. 101

[Che06] J.H. Cheon. Security analysis of the strong Diffie-Hellman problem. In *Advances in Cryptology—EUROCRYPT'06*, pages 1–13, 2006. 15

[CHL$^+$05] M. Chase, A. Healy, A. Lysyanskaya, T. Malkin, and L. Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *Advances in Cryptology—EUROCRYPT'05*, pages 422–439, 2005. 24

[CJZ05] X. Chunxiang, Z. Junhui, and Q. Zhiguang. A Note on Secure Key Issuing in ID-based Cryptography. Cryptology ePrint Archive, Report 2005/180, 2005. `http://eprint.iacr.org/2005/180`. 77

[CKAS02] C. Cachin, K. Kursawe, A.Lysyanskaya, and R. Strobl. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *ACM CCS'02*, pages 88–97, 2002. 4, 11, 40, 44, 48, 50, 51, 57, 64, 66, 71, 136

[CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In *Advances in Cryptology—CRYPTO'01*, pages 524–541, 2001. 51

[CKS00] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography. In *ACM PODC'00*, pages 123–132, 2000. 2, 51

[CL01] M. Castro and B. Liskov. Byzantine Fault Tolerance Can Be Fast. In *IEEE DSN'01*, pages 513–518, 2001. 141

[CL02] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst. (TOCS)*, 20(4):398–461, 2002. 25, 42, 51, 134, 141, 143

[CL05] J. Camenisch and A. Lysyanskaya. A Formal Treatment of Onion Routing. In *Advances in Cryptology—CRYPTO'05*, pages 169–187, 2005. 102, 104, 120, 122, 123

[CL06] H. Chien and R. Lin. Identity-based Key Agreement Protocol for Mobile Ad-hoc Networks Using Bilinear Pairing. In *IEEE SUTC'06*, pages 520–529, 2006. 111

[CML$^+$99] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance. In *OSDI'99*, pages 177–190, 1999. 141

[Cor00] J.-S. Coron. On the Exact Security of Full Domain Hash. In *Advances in Cryptology—CRYPTO'00*, pages 229–235, 2000. 108

[CP92] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology—CRYPTO'92*, pages 89–105, 1992. 18

[CP02] C. Cachin and J. Poritz. Secure Intrusion-tolerant Replication on the Internet. In *IEEE DSN'02*, pages 167–176, 2002. 141

[CR93] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *ACM STOC'93*, pages 42–51, 1993. 40, 51

[CS97] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms, 1997. Technical Report No. 260, Dept. of Computer Science, ETH Zurich. 17

[Dai98] W. Dai. PipeNet 1.1. `www.weidai.com/pipenet.txt`, 1998. Accessed Nov. 2009. 102

[Dam99] I. Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark*, volume 1561 of *LNCS*, pages 63–86, 1999. 17

[DDM03] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Symposium on Security and Privacy 2003*, pages 2–15, 2003. 120

[DE06] R. Dupont and A. Enge. Provably Secure Non-Interactive Key Distribution Based on Pairings. *Discrete Applied Mathematics*, 154(2):270–276, 2006. 108

[DF89] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Advances in Cryptology—CRYPTO'89*, pages 307–315, 1989. 1, 2, 11

[DG09] G. Danezis and I. Goldberg. Sphinx: A Compact and Provably Secure Mix Format. In *IEEE Symposium on Security and Privacy 2009*, pages 269–282, 2009. 6, 102, 120, 121, 122

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22(6):644–654, 1976. 14

[DL04] G. Danezis and B. Laurie. Minx: A Simple and Efficient Anonymous Packet Format. In *WPES'04*, pages 59–65, 2004. 120

[DLS88] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. of ACM*, 35(2):288–323, 1988. 42

[DLS$^+$04] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *USENIX NSDI'04*, pages 85–98, 2004. 148

[DM08] R. Dingledine and N. Mathewson. Tor Protocol Specification. `https://www.torproject.org/svn/trunk/doc/spec/tor-spec.txt`, 2008. Accessed January 2009. 124, 125

[DMS04] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security'04*, pages 303–320, 2004. 5, 102, 103, 112

[Dou02] J. R. Douceur. The Sybil Attack. In *IPTPS '02*, pages 251–260, 2002. 135

[DvOW92] W. Diffie, P.C. van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992. 65

[EDG09] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *USENIX Security'09*, pages 33–50, 2009. 126

[Fel87] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *IEEE FOCS'87*, pages 427–437, 1987. 4, 11, 14, 34, 40

[FLP85] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. of ACM*, 32(2):374–382, 1985. 42

[FM02] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *ACM CCS'02*, pages 193–206. ACM, 2002. 102

[FMY99] Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptively-secure distributed public-key systems. In *ESA'99*, pages 4–27, 1999. 11

[FO99] E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Advances in Cryptology—CRYPTO'99*, pages 537–554, 1999. 86, 87, 88, 91

[FPJ$^+$07] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a Million User DHT. In *IMC'07*, pages 129 – 134, 2007. 139

[FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194, 1986. 4, 17

[FS02] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *ACM-SIAM SODA'02*, pages 94–103, 2002. 142

[FSY05] A. Fiat, J. Saia, and M. Young. Making Chord Robust to Byzantine Attacks. In *ESA'05*, pages 803–814, 2005. 139, 140, 142, 143

[Gal05] D. Galindo. Boneh-Franklin Identity Based Encryption Revisited. In *ICALP'99*, pages 791–802, 2005. 88

[GDJ06] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS'06*, 2006. 149, 150

[GDS$^+$03] P. K. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *SOSP'03*, pages 314–329, 2003. 149

[GGDS07] R. Gangishetti, M. Choudary Gorantla, M. Das, and A. Saxena. Threshold key issuing in identity-based cryptosystems. *Computer Standards & Interfaces*, 29(2):260–264, 2007. 77

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. In *Advances in Cryptology—EUROCRYPT'96*, pages 354–371, 1996. 143, 144

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT'99*, pages 295–310, 1999. 11, 13, 76

[GJKR01] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. *Inf. Comput.*, 164(1):54–84, 2001. 39

[GJKR03] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol. In *CT-RSA'03*, pages 373–390, 2003. 11, 13

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. of Cryptology*, 20(1):51–83, 2007. 4, 11, 12, 13, 14, 36, 39, 57, 59, 97

[GKLL09] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *USENIX Security'09*, pages 299–315, 2009. 139

[GMC07] F. Guo, Y. Mu, and Z. Chen. Identity-Based Encryption: How to Decrypt Multiple Ciphertexts Using a Single Decryption Key. In *Pairing'07*, pages 392–406, 2007. 26

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 44

[Gol06] I. Goldberg. On the Security of the Tor Authentication Protocol. In *PET'06*, pages 316–331, 2006. 103, 123

[Goy07] V. Goyal. Reducing Trust in the PKG in Identity Based Cryptosystems. In *Advances in Cryptology—CRYPTO'07*, pages 430–447, 2007. 24, 25, 26, 77

[GPS08] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008. 5, 16, 78, 86

[GRR98] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *ACM PODC'98*, pages 101–111, 1998. 34, 79, 80

[GRS96] D. M. Goldschlag, M. Reed, and P. Syverson. Hiding Routing Information. In *IH'96*, pages 137–150, 1996. 103, 112

[GS09] M. Geisler and N. P. Smart. Distributing the Key Distribution Centre in Sakai-Kasahara Based Systems. In *IMA Int. Conf. on Cryptography and Coding'09*, pages 252–262, 2009. 77

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In *Advances in Cryptology—CRYPTO'95*, pages 339–352, 1995. 11, 28, 63, 64, 71

[HK04] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-peer Networks. In *DISC'04*, pages 321–336, 2004. 139, 142

[HMV04] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, Inc., 2004. 9

[HT93] V. Hadzilacos and S. Toueg. Fault-tolerant Broadcasts and Related Problems. In *Distributed systems (2nd Ed.)*, pages 97–145. ACM Press, 1993. 68

[Hua07] D. Huang. Pseudonym-Based Cryptography for Anonymous Communications in Mobile Ad Hoc Networks. *International Journal of Security and Networks*, 2(3–4):272–283, 2007. 104

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *IEEE GLOBALCOM'87*, pages 99–102, 1987. 41

[JAvR06] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In *OSR'06*, pages 3–13, 2006. 139

[JL00] S. Jarecki and A. Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In *Advances in Cryptology—EUROCRYPT'00*, pages 221–242, 2000. 11

[JN03] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. *J. of Cryptology*, 16(4):239–247, 2003. 19, 79, 80

[JN08] M. Joye and G. Neven. *Identity-Based Cryptography - Volume 2 Cryptology and Information Security Series*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008. 75

[Jou02] A. Joux. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *ANTS-V*, pages 20–32, 2002. 93

[KBC$^+$00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, W. Weimer H. Weatherspoon, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *ASPLOS'00*, pages 190–201, 2000. 141

[KG09] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In *IEEE ICDCS'09*, pages 119–128, 2009. 4, 11, 39

[KG10a] A. Kate and I. Goldberg. Distributed private-key generators for identity-based cryptography. To appear at SCN'10, 2010. Available as Cryptology ePrint Archive, Report 2009/355. 4

[KG10b] A. Kate and I. Goldberg. Using Sphinx to Improve Onion Routing Circuit Construction. In *FC'10*, 2010. An extended version is available at `http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-33.pdf`. 6, 102

[KKA03] A. Khalili, J. Katz, and W. Arbaugh. Toward Secure Key Distribution in Truly Ad-Hoc Networks. In *IEEE Workshop on Security and Assurance in Ad-Hoc Networks 2003*, pages 342–346, 2003. 2, 39, 76, 111

[KM05] N. Koblitz and A. Menezes. Pairing-Based Cryptography at High Security Levels. In *10th IMA Int. Conf. on Cryptography and Coding*, pages 13–36, 2005. 124

[KT08] A. Kapadia and N. Triandopoulos. Halo: High-Assurance Locate for Distributed Hash Tables. In *NDSS'08*, 2008. 142

[KZG07] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *PETS'07*, pages 95–112, 2007. 2, 5, 39, 76, 101, 102, 121

[KZG09] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing with Improved Forward Secrecy. *To appear in ACM TISSEC*, 2009. 5, 102

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Polynomial Commitments. Technical Report CACR 2010-10, Centre for Applied Cryptographic Research, University of Waterloo, 2010. In Submission. 3, 11, 23, 30

[KZH07] A. Kate, G. M. Zaverucha, and U. Hengartner. Anonymity and Security in Delay Tolerant Networks. In *SecureComm'07*, pages 504–513, 2007. 111

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. 131

[LBD$^+$04] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Secure key issuing in ID-based cryptography. In *ACSW Frontiers'04*, pages 69–74, 2004. 77

[LNBK02] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *ACM PODC'02*, pages 233–242, 2002. 149

[LSB$^+$05] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring bandwidth between planetlab nodes. In *PAM'05*, pages 292–305, 2005. 133

[LSP82] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. 4

[LY10] B. Libert and M. Yung. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *TCC'10*, pages 499–517, 2010. 24

[Lyn09] B. Lynn. PBC Library. `http://crypto.stanford.edu/pbc/`, 2009. Accessed April 2009. 124, 126, 127, 132

[MCPS03] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol—Version 2. IETF Internet Draft, 2003. 120

[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 August 2009. 132

[Möl03] B. Möller. Provably Secure Public-Key Encryption for Length-Preserving Chaumian Mixes. In *CT-RSA'03*, pages 244–262, 2003. 102, 120

[MOV91] A. Menezes, T. Okamoto, and S. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *ACM STOC'91*, pages 80–89, 1991. 124

[MOV97] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1st edition, 1997. 9, 14, 24, 103, 120, 121

[MSK02] S. Mitsunari, R. Sakai, and M. Kasahara. A New Traitor Tracing. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E85-A(2):481–484, 2002. 15

[MVdV04] S. Mauw, J. Verschuren, and E. de Vink. A Formalization of Anonymity and Onion Routing. In *ESORICS'04*, pages 109–124, 2004. 102

[MW08] S. J. Murdoch and R. N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Systems. In *PETS'08*, pages 115–132, 2008. 127

[Ngu05] L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA'05*, pages 275–292, 2005. 25, 26

[Nie02] J. B. Nielsen. A Threshold Pseudorandom Function Construction and Its Applications. In *Advances in Cryptology—CRYPTO'02*, pages 401–416, 2002. 2, 39

[NPR99] M. Naor, B. Pinkas, and O. Reingold. Distributed Pseudo-random Functions and KDCs. In *Advances in Cryptology—EUROCRYPT'99*, pages 327–346, 1999. 2, 39

[NW03] M. Naor and U. Wieder. A Simple Fault Tolerant Distributed Hash Table. In *IPTPS'03*, pages 88–97, 2003. 139, 142, 143

[NW06] A. Nambiar and M. Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *ACM CCS'06*, pages 17–26, 2006. 142, 149

[OO05] E. Okamoto and T. Okamoto. Cryptosystems Based on Elliptic Curve Pairing. In *MDAI'05*, pages 13–23, 2005. 104

[ØS07] L. Øverlier and P. Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *PETS'07*, pages 134–152, 2007. 102, 103, 115, 116, 119

[OY91] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks (Ext. Abstract). In *ACM PODC'91*, pages 51–59, 1991. 63

[PACR03] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003. 4, 131

[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. 40

[Ped91a] T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Eurocrypt'91*, pages 522–526. Springer-Verlag, 1991. 11, 40

[Ped91b] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology—CRYPTO'91*, pages 129–140, 1991. 2, 4, 11, 12, 13, 17, 34, 59

[Pip76] N. Pippenger. On the evaluation of powers and related problems. In *IEEE SFCS(FOCS)'76*, pages 258–263, 1976. 34

[Pla07] PlanetLab Hardware Requirements. `http://www.planet-lab.org/hardware`, 2007. Accessed May 2010. 133

[PMTZ06] V. Pappas, D. Massey, A. Terzis, and L. Zhang. A Comparative Study of the DNS Design with DHT-Based Alternatives. In *INFOCOM'06*, pages 1–13, 2006. 139

[Rei95] M. K. Reiter. The Rampart Toolkit for Building High-Integrity Services. In *Intl. Workshop on Theory and Practice in Distributed Systems*, pages 99–110, 1995. 141

[RGRK04] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In USENIX *Annual Technical Conf.*, pages 127–140, 2004. 149

[RIO^{+}06] S. Rahman, A. Inomata, T. Okamoto, M. Mambo, and E. Okamoto. Anonymous Secure Communication in Wireless Mobile Ad-hoc Networks. In *ICUCT'06*, pages 140–149, 2006. 104

[RKB07] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live. In *HotDep'07*, 2007. 141

[RL03] R. Rodrigues and B. Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. Technical Report TR/932, MIT LCS, 2003. 141

[RLS02] R. Rodrigues, B. Liskov, and L. Shrira. The Design of a Robust Peer-to-Peer System. In *ACM SIGOPS European Workshop 2002*, pages 117–124, 2002. 141

[RP02] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *WPES'02*, pages 91–102, 2002. 102

[RSG98] M. Reed, P. Syverson, and D. Goldschlag. Anonymous Connections and Onion Routing. *IEEE J-SAC*, 16(4):482–494, 1998. 101, 102, 103

[SB08] R. Snader and N. Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *NDSS'08*, 2008. 127

[Sch90] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990. 131, 141

[Sch91] C. P. Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptology*, 4(3):161–174, 1991. 17

[SENB07] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *IMC'07*, pages 117–122, 2007. 139

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *MMCN'02*, pages 314–329, 2002. 149, 150

[Sha79] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979. 1, 10, 80

[Sha84] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology—CRYPTO'84*, pages 47–53, 1984. 75

[Sho00] V. Shoup. Practical Threshold Signatures. In *Advances in Cryptology—EUROCRYPT'00*, pages 207–220, 2000. 143, 144

[Sho06] V. Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, 1st edition, 2006. 29

[SK03] R. Sakai and M. Kasahara. ID based Cryptosystems with Pairing on Elliptic Curve. Cryptology ePrint Archive, Report 2003/054, 2003. 5, 77, 78, 83, 89

[SK05] A. Seth and S. Keshav. Practical Security for Disconnected Nodes. In *IEEE ICNP NPSec 2005*, pages 31–36, 2005. 111

[SLL08] D. A. Schultz, B. Liskov, and M. Liskov. Mobile Proactive Secret Sharing. In *ACM PODC'08*, page 458, 2008. (Extended Draft). 40

[SM02] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *First Intl. Workshop on Peer-to-Peer Systems*, pages 261–269, 2002. 139

[SMK$^+$01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM'01*, pages 149–160, 2001. 140

[SOK00] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security (SCIS 2000)*, 2000. 101, 104

[SSH08] E. Shimshock, M. Staats, and N. Hopper. Breaking and Provably Fixing Minx. In *PETS'08*, pages 99–114, 2008. 120

[Sti05] D. R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Inc., 3rd edition, 2005. 10

[STRL00] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114, 2000. 102, 103

[SY08] J. Saia and M. Young. Reducing Communication Costs in Robust Peer-to-Peer Networks. *Information Processing Letters*, 106(4):152–158, 2008. 6, 139, 140, 142

[Tor10] Tor: Anonymity Online. `https://www.torproject.org`, 2010. Accessed May 2010. 102

[Wal02] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *ISSS'02*, pages 253–258, 2002. 139

[Wat05] B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology—EUROCRYPT'05*, pages 114–127, 2005. 99

[WZF05] H. Wang, Y. Zhang, and D. Feng. Short Threshold Signature Schemes Without Random Oracles. In *INDOCRYPT'03*, pages 297–310, 2005. 77

[YKGK10] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. To appear at ICDCS'10, 2010. An extended version available as CACR 2009-31 at `http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-31.pdf`. 6, 140, 146, 147

[ZSvR05] L. Zhou, F. B. Schneider, and R. van Renesse. APSS: Proactive Secret Sharing in Asynchronous Systems. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 8(3):259–286, 2005. 40