Asynchronous Distributed Private-Key Generators for Identity-Based Cryptography

Aniket Kate Ian Goldberg
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
{akate,iang}@cs.uwaterloo.ca

Abstract

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 private-key generator (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 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.

In this paper, we design distributed PKG setup and private key extraction protocols in an asynchronous communication model for three important IBE schemes; namely, Boneh and Franklin's IBE, Sakai and Kasahara's IBE, and Boneh and Boyen's BB_1-IBE. We give special attention to the applicability of our protocols to all possible types of bilinear pairings and prove their IND-ID-CCA security in the random oracle model. Finally, we also perform a comparative analysis of these protocols and present recommendations for their use.

1 Introduction

In 1984, Shamir [58] 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 [10] 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 (for details, refer a recent book on IBC [39] and references therein).

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 the 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 certifying 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),
- the PKG can make clients' private keys public without any possible detection, and
- in a validity period-based key revocation system [10], 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 [10] 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 [10] propose *verifiable secret sharing* (VSS) of the master key among multiple PKGs using Shamir secret sharing with a *dealer* [57] to design a distributed PKG and also hint towards a completely distributed approach using the distributed (shared) key generation (DKG) schemes of Gennaro et al. [31]; however, they do not provide a security proof. Further, none of the IBE schemes defined after [10] consider the design of a distributed PKG.

On the system side, the DKG schemes [31] suggested in [10] to design a distributed PKG are not advisable for use over the Internet. These DKG schemes are defined for the *synchronous communication model*, having bounded message delivery delays and processor speeds, and do not provide *safety* (the protocol does not fail or produce incorrect results) and *liveness* (the protocol eventually terminates) over the asynchronous Internet, having no bounds on message transfer delays or processor speeds.

As a whole, although various proposed practical applications using IBE, such as key distribution in ad-hoc networks [42], pairing-based onion routing [41] or verifiable random functions from identity-based key encapsulation [1], require 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 for distributed PKGs for IBC schemes that can function over the Internet forms the motivation of this work.

Contributions. We present asynchronous distributed PKGs for all three important IBE frameworks: namely, full-domain-hash IBEs, exponent-inversion IBEs and commutative-blinding IBEs [12]. We propose distributed PKG setups and distributed private-key extraction protocols for Boneh and Franklin's IBE (BF-IBE) [10], Sakai and Kasahara's IBE (SK-IBE) [54], and Boneh and Boyen's (modified) BB$_1$-IBE [13, 12] 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. In terms of feasibility, we ensure that our protocols work for all three pairing types defined by Galbraith et. al. [28].

We prove the IND-ID-CCA security of the defined three 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.

In the process, we also design practical asynchronous protocols for distributed multiplication and inverse computation tasks, which have their own applications. Here, asynchrony is obtained using an asynchronous DKG in the appropriate fashion. Further, we replace proofs of knowledge by comparatively cheaper pairing-based verification.

Observing that a distributed (shared) key generator (DKG) is the single most important component of distributed PKG, we implement a recently devised asynchronous DKG protocol [40] and demonstrate its efficiency and reliability with extensive testing over the PlanetLab platform [52]. Finally, using operation counts, key sizes, and possible pairing types, we compare the performance of the distributed PKGs we define and also briefly discuss the proactive security and group modification primitives for them.

In §2, we compare various techniques suggested to solve the key escrow and single point of failure problems in IBC. We also discuss previous work related to DKG protocols. In §3, we describe a realistic asynchronous system model over the Internet and justify the choices made, while we define and describe cryptographic tools in our model in §4. With this background, in §5, we define and prove distributed PKG protocols for the BF-IBE, SK-IBE and BB_1-IBE schemes. We then implement a practical DKG protocol, and test its performance over the PlanetLab platform in §6. We also compare the IBE schemes based on their distributed PKGs and touch upon proactive security and group modification protocols for the system.

2 Related Work

We divide the related work into two parts. Distributed (shared) key generation is the most important component for distributed private-key generation in identity-based cryptography. We first discuss the existing work towards distributed key generation. As designing distributed PKGs is our main goal in this work, we concentrate on protocols in computational (as opposed to unconditional / information-theoretic) settings. Although somewhat ignored, there have been some efforts to mitigate the single point of failure and the key escrow issues in IBC systems; in the latter part of this section, we compare these alternatives with distributed PKG.

Although we are defining protocols for IBE schemes, as we are concentrating on distributed cryptographic protocols and due to space constraints, we do not include a comprehensive account of IBE here. We refer readers to [12] 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.

Distributed Key Generation. The notion of secret sharing was introduced independently by Shamir [57] and Blakley [7] in 1979. Since then, it has remained an important topic in security research. Significantly, Chor et al. [21] introduced verifiability in secret sharing. Feldman [24] developed the first efficient and non-interactive VSS protocol and Pedersen [50] presented a modification to it. However, these VSS are defined assuming a synchronous communication model. For an *asynchronous communication model*, Cachin et al. (AVSS) [14], Zhou et al. (APSS) [62], and Schultz et al. (MPSS) [56] defined VSS schemes in the computational setting. Of these, the APSS protocol is impractical for any reasonable system size, as it has an exponential $\binom{n}{t}$ factor in the message complexity (number of messages transferred), while MPSS is developed for a more mobile setting where set of the system nodes has to change completely between two consecutive phases. AVSS by Cachin et al. with its seemingly optimal *communication complexity* (number of bits transferred) is certainly a suitable choice for a distributed PKG system.

Pedersen [51] introduced the concept of distributed key generation and developed a DKG, where each node runs a variation of Feldman's VSS and distributed shares are added at the end to generate a combined shared secret without a dealer. Gennaro et al. [31] presented a simplification using just the original Feldman VSS called the Joint Feldman DKG (JF-DKG). Further, they found that DKGs based on the Feldman VSS (or using Feldman commitments [24]) do not guarantee uniformly random secret keys and define a new DKG combining Feldman and Pedersen commitments [51] which increases the *latency* (number of communication rounds) by one. However, in [32], they observed that DKGs based on Feldman commitments produce hard instances of discrete logarithm problems (DLPs), which may be sufficient for the security of some threshold cryptographic schemes.

To the best of our knowledge, the first DKG scheme in an asynchronous setting was only defined recently by Kate and Goldberg [40]. This protocol modifies the AVSS protocol to a more realistic hybrid model and performs leader-based agreement with a leader-changing mechanism to decide which of the nodes' VSS will be included in the DKG calculation; that is, whereas in synchronous DKG schemes such as Pedersen's above, all of the successful VSSs can be added at the end of the protocol to determine the final master key shares, in the asynchronous setting, some global consensus must be reached in order to find a sufficiently large set of VSSs which all honest nodes have completed. We implement this DKG protocol and verify its efficiency and reliability. Consequently, this DKG system forms the basis of our distributed PKG protocols. The original asynchronous DKG protocol uses Feldman commitments and consequently does not guarantee uniform randomness of the key. However, we observe that, in the random oracle model, using non-interactive zero-knowledge proofs of knowledge based on the Fiat-Shamir methodology [25], if required, it is possible to achieve uniform randomness in their scheme. In such a scheme, Feldman commitments are initially replaced by Pedersen commitments; the Feldman commitments are introduced only at the end of the protocol to obtain the required private key. The zero-knowledge proofs are used to show that the Feldman and Pedersen commitments both commit to the same values.

All of the above schemes are proved secure only against a static adversary, which can only choose its t compromisable nodes before a protocol run. They are not considered secure against an adaptive adversary because their simulation-based security proofs do not go through when the adversary can corrupt nodes adaptively.[33, §4.4] Feldman claimed [24, §9.3] that his VSS protocol is also secure against adaptive adversaries even though his simulation-based security proof did not work out. Canetti et al. [16] presented a scheme provably secure against adaptive adversaries with at least two more communication rounds as compared to JF-DKG and with interactive zero-knowledge proofs. Due to the inefficiency of adaptive (provably) secure DKG protocols, we stick to protocols provably secure only against a static adversary, though they have remained unattacked by an adaptive adversary for the last 22 years.

Alternatives to a Distributed PKG. Although none of the IBE schemes except BF-IBE considered distributed PKG setup and key extraction in order to solve the inherent key escrow and single point of failure issues, there have been a few other efforts in the literature to counter those.

Al-Riyami and Paterson [2] introduce *certificateless public key cryptography* (CL-PKC) to address the key escrow problem by combining IBC with public-key cryptography (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, CL-PKC becomes ineffective for systems without continuous network access, where IBC is considered to be an important tool.

Lee et al. [44] and Gangishetti et al. [30] 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. [22] for [44], 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.

Recently, Goyal [35] 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 a generic distributed PKG wanting.

Threshold versions of signature schemes obtained from some IBE schemes using the Naor transform have been proposed and proved previously [8, 59]. 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 shares it generates for a client and that private key shares are not available in public as commitments.

3 System Model and Assumptions

In this section, we briefly discuss the assumptions and the system model for our distributed PKG system, giving special attention to its practicality over the Internet. We follow the system model of [40], which closely depicts the Internet, and as their DKG forms the basis of our distributed PKGs.

3.1 Communication Model

In the theoretical sense, distributed protocols designed with a synchronous or a *partially synchronous* (bounded message delivery delays and processor speeds, but the bounds are unknown and eventual [23]) communication assumption tend to be more efficient in terms of latency and message complexity than their counterparts designed with an asynchronous communication assumption. However, protocols defined in the synchronous or partially synchronous communication model invariably use some time bounds in their definition. An adversary, knowing those bounds, may slow down the protocol by appropriately delaying its messages, which makes deciding the time bounds correctly a difficult problem to solve. On the other hand, protocols defined for the asynchronous communication model use only numbers and types of messages and guarantee to finish quickly with only honest nodes communicating promptly. Therefore, we assume an asynchronous communication model.

Weak Synchrony (only for liveness). Generating true randomness in a completely distributed (dealerless) asynchronous setting efficiently, without using a DKG, although not impossible [17], is a difficult task to perform; the known computational threshold coin-tossing algorithms [15] require a dealer or a synchronous communications assumption. As observed in [40], asynchronous DKG requires a protocol to solve the *agreement on a set* problem [5], which needs distributed randomness or a synchrony assumption [26]. In the absence of an efficient randomization procedure, [40] uses a *weak synchrony* assumption by Castro and Liskov [18] for liveness, but not safety. According to this assumption, a function $delay(t)$, defining the message transmission delay of a message sent at time t, does not grow faster than t indefinitely. Assuming that network faults are eventually repaired and DoS attacks eventually stop, this assumption is valid in practice. We further discuss this assumption in §6.1.

3.2 Hybrid Adversary Model

Instead of using a standard t-Byzantine adversary in a system with n nodes $P_1, P_2, \ldots, P_n$, we use a *hybrid* adversary introduced in [3], having another f non-Byzantine crashes, and modified in [40] to include network link failures.

For the standard t-Byzantine adversary, t nodes compromised or crashed by the adversary remain compromised forever. This does not depict the adversary model over the Internet accurately. Along with arbitrary behaviour by t Byzantine nodes, some nodes can just crash silently without showing malicious behaviour or just get disconnected from the system due to network failure or partitioning. As the adversary does not capture these f nodes or their secret parameters, it is not computationally and communicationally optimal to consider these nodes as Byzantine. It also gives rise to a sub-optimal resilience of $n \geq 3(t+f)+1$ instead of the $n \geq 3t+2f+1$ bound effected by treating crashes and link failures separately from the Byzantine adversary.

In this *hybrid adversary model*, crashes and link failures belong to the same set of f nodes, as from a perspective of any other node of the system a crashed node behaves exactly same as a node whose link with it is broken. We recover secrets at these f nodes immediately after their trusted rebooting, which gives us the assumption that all non-Byzantine nodes may crash and recover repeatedly with a maximum f crashed nodes at any instant. If two nodes cannot communicate, then we treat at least one of two nodes as being either Byzantine or one of the currently crashed nodes. That is, following the standard asynchronous communication model literature, we assume that the adversary controls the network, but faithfully delivers all the messages between two honest uncrashed nodes.

3.3 Cryptographic Background

Bilinear Pairings. IBC extensively utilizes bilinear pairings over elliptic curves. For three cyclic groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ (all of which we shall write multiplicatively) of the same prime order p, a *bilinear pairing* e is a map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ with 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 and come back to $\mathbb{G}$ or $\hat{\mathbb{G}}$. All pairings considered in this paper are admissible and infeasible to invert. We call such groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ *pairing-friendly* groups. We refer readers to [6, Chap. IX and X] for a detailed mathematical discussion of bilinear pairings.

Following [28], we consider three types of pairings: namely, type 1, 2, and 3. In *type* 1 pairings, an isomorphism $\phi : \hat{\mathbb{G}} \to \mathbb{G}$ as well as its inverse ϕ^{-1} are efficiently computable. These are also called *symmetric pairings* as for such pairings $e(g, \hat{g}) = e(\phi(\hat{g}), \phi^{-1}(g))$ for any $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$, and we usually just identify $\mathbb{G}$ with $\hat{\mathbb{G}}$ in this case. 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 a survey by Galbraith et al. [28].

Cryptographic Assumptions. As mentioned in §2, for efficiency reasons, we assume 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 κ-bit prime. For commitments and proofs of knowledge, we use the *discrete logarithm* (DLog) [47, Chap. 3] assumption.

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 such that the adversary has to perform 2^κ operations to break the system. For the security of the IBE schemes, we use the *bilinear Diffie-Hellman* (BDH) [36] and *bilinear Diffie-Hellman inversion* (BDHI) [48, 9] assumptions. Here, we recall the definitions of generic versions (for asymmetric pairings) of these two assumptions from [12]. Note that a function $\epsilon(\cdot)$ is called *negligible* if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$.

BDH Assumption: Given a tuple $(g, \hat{g}, g^a, \hat{g}^a, g^b, \hat{g}^c)$ in a bilinear group $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T \rangle$, the BDH problem is a problem to compute $e(g, \hat{g})^{abc}$. The BDH assumption then states that it is infeasible to solve a random instance of the BDH problem, with non-negligible probability, in time polynomial in the size of the problem instance description.

BDHI Assumption: Given two tuples $(g, g^x, g^{(x^2)}, \ldots, g^{(x^q)})$ and $(\hat{g}, \hat{g}^x, \hat{g}^{(x^2)}, \ldots, \hat{g}^{(x^q)})$ in a bilinear group $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T \rangle$, the q-BDHI problem is a problem to compute $e(g, \hat{g})^{1/x}$. The BDHI assumption for some polynomially bounded q states that it is infeasible to solve a random instance of the q-BDHI problem, with non-negligible probability, in time polynomial in the size of the problem instance description.

4 Cryptographic Tools

In this section, we describe important cryptographic tools required to design distributed PKGs in the 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. Note that these tools are also useful in other asynchronous computational multiparty settings.

4.1 Homomorphic Commitments over $\mathbb{Z}_p$

A verification mechanism for a consistent dealing is fundamental to VSS. It is achieved using distributed computing techniques in the unconditional setting. In the computational setting, *homomorphic commitments* provide an efficient alternative. Let $\mathcal{C}(\alpha, [r]) \in \mathbb{G}$ be a homomorphic commitment to $\alpha \in \mathbb{Z}_p$, where r is an optional randomness parameter and $\mathbb{G}$ is a (multiplicative) group. For such a homomorphic commitment, given $C_1 = \mathcal{C}(\alpha_1, [r_1])$ and $C_2 = \mathcal{C}(\alpha_2, [r_2])$, we have $C_1 \cdot C_2 = \mathcal{C}(\alpha_1 + \alpha_2, [r])$.

VSS protocols utilize two forms of commitments. Let g and h be two random generators of $\mathbb{G}$. Feldman, for his VSS protocol [24], used a commitment scheme of the form $\mathcal{C}_{\langle g \rangle}(\alpha) = g^\alpha$ with computational security under the DLog assumption and unconditional share integrity. Pedersen [51] presented another commitment of the form $\mathcal{C}_{\langle g,h \rangle}(\alpha, r) = g^\alpha h^r$ with unconditional security but computational integrity under the DLog assumption. In PKC based on computational assumptions, with adversarial access to the public key, unconditional security of the secret (private key or master key) is impossible. Further, in VSS schemes based on Pedersen commitments, in order to randomly select the generator h, an additional round of communication is required during bootstrapping. Consequently, in our scheme, we use simple and efficient Feldman commitments, except during a special case described in the DKG discussion below.

In their VSSs, Feldman and Pedersen use commitments of *coefficients* of shared polynomials. However, following the computational multiparty computation protocol by Gennaro et al. [34] and AVSS by Cachin et al. [14], we instead use commitments of *evaluations* of shared polynomials. This reduces the communication complexity (the total bit length of messages exchanged) of AVSS by a linear factor and makes verifications of shares' products easier in the distributed multiplication protocol of [34]. To that end, we define the *Feldman commitment vector* $\mathcal{C}^{(s)}_{\langle g \rangle} = [g^s, g^{\varphi(1)}, \cdots, g^{\varphi(n)}]$ where φ is a randomly selected polynomial of degree t over $\mathbb{Z}_p$ with $\varphi(0) = s$. Similarly, the *Pedersen commitment vector* $\mathcal{C}^{(s,s')}_{\langle g,h \rangle} = [g^s h^{s'}, g^{\varphi(1)} h^{\psi(1)}, \cdots, g^{\varphi(n)} h^{\psi(n)}]$ where φ is as above, and ψ is similar, but with $\psi(0) = s'$. The j^{th} element of a Feldman commitment vector (counting from 0) will be denoted by $\left(\mathcal{C}^{(s)}_{\langle g \rangle}\right)_j$ (and similarly for Pedersen commitment vectors).

4.2 Non-interactive Proofs of Knowledge

As we assume the random oracle model in this paper, we can use non-interactive zero-knowledge proofs of knowledge (NIZKPK) based on the Fiat-Shamir methodology [25]. In particular, we use a variant of NIZKPK of a discrete logarithm and one for proof of equality of two discrete logarithms.

We employ a variant of NIZKPK of a discrete logarithm where given a Feldman commitment $\mathcal{C}_{\langle g\rangle}(s)$ and a Pedersen commitment $\mathcal{C}_{\langle g,h\rangle}(s,r)$ for $s,r \in \mathbb{Z}_p$, a prover proves that she knows s and r such that $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h\rangle}(s,r) = g^s h^r$. That is, the prover proves that the Feldman commitment and the Pedersen commitment are to the same value s. We denote this proof as

$$\text{NIZKPK}_{\equiv Com}(s, r, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r)) = \pi_{\equiv Com} \in \mathbb{Z}_p^3. \tag{1}$$

We describe it in detail in Appendix A; it is nearly equivalent to proving knowledge of two discrete logarithms separately.

We also use another NIZKPK (proof of equality) of discrete logarithms [19] such that given two Feldman commitments $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle h\rangle}(s) = h^s$, a prover proves equality of the associated discrete logarithms. We denote this proof as

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

and refer readers to Appendix A for details. Note that g and h can belong two different groups of the same order.

There exists an easier way to prove this equality of discrete logarithms if a pairing between the groups generated by g and h is available. Using a technique due to Joux and Nguyen [38] to solve the DDH problem over pairing-friendly groups, given g^x and $h^{x'}$ the verifier checks if $e(g, h^{x'}) \stackrel{?}{=} e(g^x, h)$. 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 (say $\mathbb{G}$ without loss of generality), then the pairing-based verification scheme does not work. In such a situation, the above NIZKPK provides a less efficient but completely practical alternative.

4.3 DKG over $\mathbb{Z}_p$

In an (n,t)-*DKG* protocol over $\mathbb{Z}_p$, a set of n nodes generates an element $s \in \mathbb{Z}_p$ in a distributed fashion with its shares $s_i \in \mathbb{Z}_p$ spread over the n nodes such that any subset of size greater than a threshold t can reveal or use the shared secret, while smaller subsets cannot. A DKG protocol consists of a *sharing* (DKG-Sh) phase and a *reconstruction* (DKG-Rec) phase. In the DKG-Sh phase, a distributed secret $s \in \mathbb{Z}_p$ is generated among n nodes such that each node P_i holds a share s_i and a commitment vector $\mathcal{C}^{(s)}$ of s and all of its shares. During the DKG-Rec phase, each node P_i reveals its share s_i and reconstructs s using verified revealed shares.

Definition 4.1. *For 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, We use a DKG protocol defined in [40] satisfying the following conditions:*

Liveness: *Once protocol DKG-Sh starts, all honest finally up nodes complete the protocol, except with negligible probability.*

Agreement: *If some honest node completes protocol DKG-Sh then, except with negligible probability, all honest finally up nodes will eventually complete protocol DKG-Sh.*

Consistency: *Once an honest node completes protocol DKG-Sh then there exists a fixed value $s \in \mathbb{Z}_p$ such that, if an honest node P_i reconstructs $z_i \in \mathbb{Z}_p$ during DKG-Rec, then $z_i = s$.*

Privacy: *If no honest node has started protocol* $\mathsf{DKG\text{-}Rec}$ *then, except with negligible probability, an adversary cannot compute the shared secret* s*.*

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

A closer look at the privacy property suggests that in the presence of an adversary, the shared secret in the above DKG may not be *uniformly* random; this is a direct effect of using only Feldman commitments.[33, §3] However, 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. Most of the schemes in this paper similarly only require the assumption that it is infeasible to compute the secret given public parameters and we stick with Feldman commitments those cases. However, we do indeed need a uniformly random shared secret in few protocols In that case, we use Pedersen commitments, but we do not employ the methodology defined by Gennaro et al. [33], which increases the latency in the system. We observe instead that with the random oracle assumption at our disposal, the communicationally demanding technique by Gennaro et al. can be replaced with the much simpler computational non-interactive zero-knowledge proof of equality of committed values $\text{NIZKPK}_{\equiv Com}$ described in Eq. 1.

We represent DKG protocols using Feldman commitments and Pedersen commitments as $\mathsf{DKG}_{\mathsf{Feld}}$ and $\mathsf{DKG}_{\mathsf{Ped}}$ respectively. For node P_i, the corresponding $\mathsf{DKG\text{-}Sh}$ and $\mathsf{DKG\text{-}Rec}$ schemes are defined as follows.

$$\left(\mathcal{C}^{(s,s')}_{\langle g,h\rangle}, [\mathcal{C}^{(s)}_{\langle g\rangle}, \text{NIZKPK}_{\equiv Com}], s_i, s'_i\right) = \mathsf{DKG\text{-}Sh}_{\mathsf{Ped}}(n,t,f,\tilde{t},g,h,\alpha_i,\alpha'_i) \quad (3)$$

$$\left(\mathcal{C}^{(s)}_{\langle g\rangle}, s_i\right) = \mathsf{DKG\text{-}Sh}_{\mathsf{Feld}}(n,t,f,\tilde{t},g,\alpha_i) \quad (4)$$

$$s = \mathsf{DKG\text{-}Rec}_{\mathsf{Ped}}(t,\mathcal{C}^{(s,s')}_{\langle g,h\rangle}, s_i, s'_i) \quad (5)$$

$$s = \mathsf{DKG\text{-}Rec}_{\mathsf{Feld}}(t,\mathcal{C}^{(s)}_{\langle g\rangle}, s_i) \quad (6)$$

Here, $\tilde{t}$ is the number of VSS instances to be chosen ($t < \tilde{t} \leq 2t+1$), $g,h \in \mathbb{G}$ are commitment generators, $\alpha_i, \alpha'_i \in \mathbb{Z}_p$ are respectively a secret and randomness shared by P_i, and $\mathcal{C}^{(s)}_{\langle g\rangle}$ and $\mathcal{C}^{(s,s')}_{\langle g,h\rangle}$ are respectively the Feldman and Pedersen commitment vectors described in §4.1. 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. (The polynomial φ for the two types of commitments will be the same in this case.)

The worst-case message and communication complexities of protocol $\mathsf{DKG\text{-}Sh}$ [40] are $O(tdn^2(n+d))$ and $O(\kappa tdn^3(n+d))$ respectively, while those of protocol $\mathsf{DKG\text{-}Rec}$ are $O(n^2)$ and $O(\kappa n^2)$ respectively. Here, the function $d(\cdot)$ bounds the number of crashes that the adversary is allowed to perform.

Distributed Random Sharing over $\mathbb{Z}_p$. This protocol generates shares of a secret z chosen jointly at random from $\mathbb{Z}_p$. Every node generates a random $r_i \in \mathbb{Z}_p$ and shares that using the $\mathsf{DKG\text{-}Sh}$ protocol with Feldman or Pedersen commitments as $\mathsf{DKG\text{-}Sh}(n,t,f,\tilde{t}=t+1,g,[h],r_i,[r'_i])$ where the generator h and randomness r'_i are only required if Pedersen commitments are used. Liveness, agreement, consistency, privacy and message and communication complexities remain the same as those of the $\mathsf{DKG\text{-}Sh}$ protocol. We represent the corresponding protocols as follows:

$$\left(\mathcal{C}^{(z)}_{\langle g\rangle}, z_i\right) = \mathsf{Random}_{\mathsf{Feld}}(n,t,f,g) \quad (7)$$

$$\left(\mathcal{C}^{(z,z')}_{\langle g,h\rangle}, [\mathcal{C}^{(z)}_{\langle g\rangle}, \text{NIZKPK}_{\equiv Com}], z_i, z'_i\right) = \mathsf{Random}_{\mathsf{Ped}}(n,t,f,g,h). \quad (8)$$

4.4 Distributed Addition over $\mathbb{Z}_p$

Let $\alpha, \beta \in \mathbb{Z}_p$ be two secrets shared among n nodes using the DKG-Sh protocol. Let polynomials $f(x), g(x) \in \mathbb{Z}_p[x]$ 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's secret sharing [57], 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 $f(x) + g(x)$ and $cf(x)$ are the respective polynomials. $f(x) + g(x)$ is random if either one of $f(x)$ or $g(x)$ is, and $cf(x)$ is random if $f(x)$ 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 $\left(\mathcal{C}^{(c\alpha)}_{\langle g\rangle}\right)_i = \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}\right)^c_i$.

4.5 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. [34, §4]. However, instead of their interactive zero-knowledge proof, we utilize a pairing-based DDH problem solving technique [38] to verify the correctness of the product value shared by a node non-interactively. For shares α_i and β_i with Feldman 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 of 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 Feldman commitments for one of the values and employ NIZKPK$_{\equiv Com}$ 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. [34] depends upon implementation efficiencies of the group operation and pairing computations.

In our IBC schemes, we always use the multiplication protocol with at least one Feldman commitment. We denote the multiplication protocol involving two Feldman commitments as $\mathsf{Mul}_{\mathsf{Feld}}$ and the one involving a combination of the two types of commitments as $\mathsf{Mul}_{\mathsf{Ped}}$. Liveness and agreement properties are exactly the same as those of DKG-Sh. For consistency, along with recoverability to a unique value (say s), protocol Mul also requires that $s = \alpha\beta$. For privacy, along with the secrecy of $\alpha\beta$ until DKG-Rec is started, the protocol should not provide any additional information about the individual values of α or β once $\alpha\beta$ is reconstructed.

$$\left(\mathcal{C}^{(\alpha\beta)}_{\langle g^*\rangle}, (\alpha\beta)_i\right) = \mathsf{Mul}_{\mathsf{Feld}}(n, t, f, g^*, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right), \left(\mathcal{C}^{(\beta)}_{\langle \hat{g}\rangle}, \beta_i\right)) \tag{9}$$

$$\left(\mathcal{C}^{(\alpha\beta,\alpha\beta')}_{\langle \hat{g},\hat{h}\rangle}, (\alpha\beta)_i, (\alpha\beta')_i\right) = \mathsf{Mul}_{\mathsf{Ped}}(n, t, f, \hat{g}, \hat{h}, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right), \left(\mathcal{C}^{(\beta,\beta')}_{\langle \hat{g},\hat{h}\rangle}, \beta_i, \beta'_i\right)) \tag{10}$$

For $\mathsf{Mul}_{\mathsf{Feld}}$, $g^* = g$ 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 DKG-Sh$(n, t, f, \tilde{t} = 2t + 1, \hat{g}, [\hat{h}], \alpha_i\beta_i, [\alpha_i\beta'_i])$ from Eq. 3 or 4. 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 the end of the DKG-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 $\alpha\beta$.

Analysis. Here, we roughly prove the properties of protocol Mul. This protocol is almost equivalent to the share renewal protocol in [40, §5.2] which is a slight modification of protocol DKG-Sh. The liveness and

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

agreement proofs are exactly the same as those of DKG-Sh [40, §4]. The basic consistency proof remains the same as that of the share renewal protocol [40, §5.2] 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 privacy 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. For ℓ shared secrets $x_1, x_2, \cdots, x_\ell$, and their corresponding Feldman 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. 9. We denote this generalization as follows.

$$\left(\mathcal{C}^{(x')}_{\langle g^*\rangle}, x'_i\right) = \mathsf{Mul}_{\mathsf{BP}}(n,t,f,g^*,\{(k_i,a_i,b_i)\}, \left(\mathcal{C}^{(x_1)}_{\langle g\rangle},(x_1)_i\right), \left(\mathcal{C}^{(x_2)}_{\langle g\rangle},(x_2)_i\right), \cdots, \left(\mathcal{C}^{(x_\ell)}_{\langle g\rangle},(x_\ell)_i\right)) \quad (11)$$

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 Feldman 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. 11 during distributed private-key extraction in the Boneh and Boyen's BB$_1$-IBE scheme in §5.5.

4.6 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-1}; this involves $O(\log p)$ distributed multiplications. However, using a technique by Bar-Ilan and Beaver [4], this can be done using just one Random, one Mul and one DKG-Rec protocol.

This protocol involves a DKG-Rec which outputs the product of the shared secret α with a distributed random element z. If z is created using Feldman 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:

$$\left(\mathcal{C}^{(\alpha^{-1})}_{\langle g^*\rangle}, (\alpha^{-1})_i\right) = \mathsf{Inverse}(n,t,f,\hat{g},\hat{h}, \left(\mathcal{C}^{(\alpha)}_{\langle g\rangle}, \alpha_i\right)) \quad (12)$$

Here g^* belongs to any group of order p. The liveness, agreement and privacy properties of the protocol are the same as those of DKG-Sh except privacy is defined in the terms of α^{-1} instead of α; for the consistency 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 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. then sends (w_i, w'_i) to each node and reconstructs $w = \textsf{DKG-Rec}_{\textsf{Ped}}(t, \mathcal{C}^{(w,w')}_{\langle \hat{g}, \hat{h} \rangle}, w_i, w'_i)$. If $w = 0$, repeats the above two steps, else locally computes $(\alpha^{-1})_i = w^{-1} z_i$;
4. finally, 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 §5.4.

Analysis. This protocol is a combination of the $\textsf{Random}_{\textsf{Ped}}$, $\textsf{Mul}_{\textsf{Ped}}$ and $\textsf{DKG-Rec}$ protocols along with some local computations. Therefore, its liveness and agreement properties follow directly from the corresponding properties of protocol $\textsf{DKG}$. Uniqueness of the recovered value follows from the consistency property of protocol $\textsf{DKG}$, while its equality to α^{-1} can be proven as follows: a share computed by a node P_i at the end of protocol $\textsf{Inverse}$ 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:

$$\textsf{DKG-Rec}_{\textsf{Feld}}(t, \mathcal{C}^{(\frac{z}{z\alpha})}_{\langle g \rangle}, \frac{z_i}{z\alpha}) = \frac{1}{z\alpha}\textsf{DKG-Rec}_{\textsf{Feld}}(t, \mathcal{C}^{(z)}_{\langle g \rangle}, z_i) = \frac{z}{z\alpha} = \alpha^{-1}$$

Privacy of protocol $\textsf{Inverse}$ follows directly from privacy of protocols $\textsf{Mul}$ and $\textsf{DKG-Sh}_{\textsf{Ped}}$. After the reconstruction of $w = z\alpha$, the distributed uniformly random element z and α remain private by the privacy properties of protocol $\textsf{Mul}$. As the final shares of α^{-1} are generated using a local computation, there is no privacy loss in the last step either. It has the same asymptotic message and communication complexities as those of protocol $\textsf{DKG-Sh}$.

5 Distributed PKG for IBE

We present and prove distributed PKG setup and private key extraction protocols for three IBE schemes: namely, Boneh and Franklin's IBE (BF-IBE) [10], Sakai and Kasahara's IBE (SK-IBE) [54], and Boneh and Boyen's IBE (BB$_1$-IBE) [12]. Each of these schemes represents a distinct important category of an IBE classification defined by Boyen [11]. 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 a recent identity-based cryptography standard (IBCS) [13] and also a comparative study by Boyen [12], which finds the above three schemes to be the most practical IBE schemes in their respective categories. In his classification, Boyen [11] 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 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.

5.1 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 compute three groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ of prime order p such that there exists a bilinear pairing e of the decided type with $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. The group order p is determined by the security parameter κ. We will write all of the groups multiplicatively.

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 $g = \phi(\hat{g})$.

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 $\mathcal{G} = \langle e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_t \rangle$.

5.2 Formal Security Model

An IBE scheme with an (n, t, f)-distributed PKG consists of the following 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 key-extraction protocol* for node P_i that takes a client identity $\texttt{ID}$, the public key vector K_{pub} and the master-secret share s_i as input and outputs a verifiable private-key share $(d_{\texttt{ID}})_i$. The client computes the private key $d_{\texttt{ID}}$ after verifying the received shares $(d_{\texttt{ID}})_i$.
- An *encryption algorithm* that takes a receiver identity $\texttt{ID}$, 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 $\texttt{ID}$ that takes a ciphertext C and the private key $d_{\texttt{ID}}$ as input and outputs a plaintext M.

Note that the above distributed PKG setup protocol doesn't 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 adaptive chosen ciphertext (IND-ID-CCA) attack, we consider the following game that a challenger plays against a polynomially bounded adversary.
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 instance. 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 it choice of the 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 $n - t - f \geq t + 1$ in any communication setting.
Phase 1: The adversary adaptively issues private-key extraction and decryption queries to the challenger. For a private-key extraction query $\langle \texttt{ID} \rangle$, the challenger simulates the distributed key extraction protocol for its $n - t$ nodes and sends verifiable private-key shares for its $n - t - f$ nodes. For a decryption query $\langle \texttt{ID}, C \rangle$, the challenger decrypts C by generating the private key $d_{\texttt{ID}}$ 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 K_{pub}, 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 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 $1/2$.

5.3 Boneh and Franklin's BF-IBE

BF-IBE [10] 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 ($\mathtt{ID}$) and s. A client with identity $\mathtt{ID}$ receives the private key $d_{\mathtt{ID}} = (H_1(\mathtt{ID}))^s = h_{\mathtt{ID}}^s \in \hat{\mathbb{G}}$, where $H_1 : \{0,1\}^* \to \hat{\mathbb{G}}^*$ 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 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 $\mathcal{C}_{\langle g \rangle}^{(s)} = [g^s, g^{s_1}, \cdots, g^{s_n}]$. We obtain this using our $\mathsf{Random}_{\mathsf{Feld}}$ protocol from Eq. 7 as

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

Private-key Extraction. After a successful setup, PKG nodes are ready to extract private keys for clients. 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 $P_i \in \mathcal{Q}$ verifies the client's identity and returns a private-key share $h_{\mathtt{ID}}^{s_i} \in \hat{\mathbb{G}}$ over a secure and authenticated channel.

2. Upon receiving $t+1$ valid shares, the client can construct her private key $d_{\mathtt{ID}}$ as $d_{\mathtt{ID}} = \prod_{P_i \in \mathcal{Q}} (h_{\mathtt{ID}}^{s_i})^{\lambda_i} \in \hat{\mathbb{G}}$, 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 $e(g, d_{\mathtt{ID}}) \overset{?}{=} e(g^s, h_{\mathtt{ID}}))$. If unsuccessful, she can verify the correctness of each received $h_{\mathtt{ID}}^{s_i}$ by checking if $e(g, h_{\mathtt{ID}}^{s_i}) \overset{?}{=} e(g^{s_i}, h_{\mathtt{ID}})$. 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 $\hat{\mathbb{G}}$ [28]; 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 ($\mathsf{FullIdent}$) [10, §4.2] secure against the BDH assumption by applying the Fujisaki-Okamoto transformation [27] to their IND-ID-CPA secure scheme ($\mathsf{BasicIdent}$). Along with $H_1 : \{0,1\}^* \rightarrow \hat{\mathbb{G}}^*$, this scheme uses three more random oracles: $H_2 : \mathbb{G}_t \rightarrow \{0,1\}^\ell$, $H_3 : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \mathbb{Z}_p$, and $H_4 : \{0,1\}^\ell \rightarrow \{0,1\}^\ell$.
Encryption: To encrypt a message M of some fixed bit length ℓ for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^\ell$, computes $r = H_3(\sigma, M)$ and $h_{\mathtt{ID}} = H_1(\mathtt{ID})$, 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 5.1. *Let 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)-$\mathsf{FullIdent}$ making at most q_E, q_D, q_{H_1}, q_{H_2}, q_{H_3}, and q_{H_4} queries. Then, there an algorithm $\mathcal{B}$ that solves the BDH 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 $O(t_1(\kappa), q_E, q_D, q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4})$.*

For their proof, Boneh and Franklin define two additional public key encryption schemes: IND-CPA secure $\mathsf{BFBasicPub}$ [10, Sec. 4.1], and its IND-CCA secure version $\mathsf{BFBasicPub}^{hy}$ [10, Sec. 4.2]. 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 $\mathsf{DKG\text{-}Sh}$ to generate $\hat{h}_{\mathtt{ID}} \in_R \hat{\mathbb{G}}$. Assuming a random oracle $H_2 : \mathbb{G} \rightarrow \{0,1\}^\ell$, where ℓ 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 $M \in \{0,1\}^\ell$, 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. Lagrange-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 [27], and random oracles $H_3 : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \mathbb{Z}_p$ and $H_4 : \{0,1\}^\ell \rightarrow \{0,1\}^\ell$.

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 [29] corrects a flaw in their proof maintaining the same proof-sequence. We also follow the same proof-sequence through Lemmas 5.1, 5.2 and 5.3 to prove Theorem 5.1:

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

Lemma 5.1. *Let 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 $\epsilon(\kappa)$ in running time $t(\kappa)$ against (n,t)-**FullIdent**. Suppose $\mathcal{A}_1$ makes at most q_E, q_D, q_{H_1}, q_{H_2}, q_{H_3}, and q_{H_4} queries. Then there is an IND-CCA adversary $\mathcal{A}_2$ that has advantage at least $\epsilon(\kappa)/q_{H_1}$ against **BFBasicPub**^{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. (Outline) The game between the challenger and the adversary $\mathcal{A}_2$ starts with the challenger running the keygen step of (n,t)-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 [29], 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 used during H_1 queries. Refer to [29, §3] for the rest of the proof. ☐

Lemma 5.2 (Fujisaki-Okamoto [27])**.** *Let H_3 and H_4 be random oracles. Let $\mathcal{A}_2$ be an IND-CCA adversary that has advantage $\epsilon_2(\kappa)$ in running time $t_2(\kappa)$ against (n,t)-**BFBasicPub**^{hy} making at most q_D, q_{H_3}, and q_{H_4} queries. Then there is 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 $t_2(\kappa)+O((q_{H_3}+q_{H_4})\ell)$, where ℓ is the message length.*

Lemma 5.3. *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 is 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 DKG-Sh, the VSSs selection may not be under $\mathcal{B}$'s control, $\mathcal{B}$ 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 $\langle \mathbb{G}_t, \{0,1\}^\ell\rangle$. 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 [10, Lemma 4.3], except during **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

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

□

5.4 Sakai and Kasahara's SK-IBE

SK-IBE [54] 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 $\texttt{ID}$ receives the private key $d_{\texttt{ID}} = \hat{g}^{\frac{1}{s+H_1'(\texttt{ID})}} \in \hat{\mathbb{G}}$, where $H_1' : \{0,1\}^* \to \mathbb{Z}_p$. Chen and Cheng [20] prove the security of SK-IBE based on the BDHIassumption.

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 Inverse protocol described in §4.6; 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 DKG-Rec. In step 4, instead of publishing, PKG nodes forward $\hat{g}^{z_i}$ and the associated NIZKPK$_{\equiv Com}$ 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 $\left(\mathcal{C}^{(r)}_{\langle\hat{g}\rangle}, r_i\right) = \mathsf{Random}_{\mathsf{Feld}}(n,t,\hat{g})$, and set $\hat{h} = \left(\mathcal{C}^{(r)}_{\langle\hat{g}\rangle}\right)_0 = \hat{g}^r$.

1. Once a client with identity $\texttt{ID}$ contacts all n nodes the system, every node P_i verifies 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 $s_i^{\texttt{ID}} = s_i + H_1'(\texttt{ID})$ and for $0 \le j \le n$, $\left(\mathcal{C}^{(s^{\texttt{ID}})}_{\langle g\rangle}\right)_j = \left(\mathcal{C}^{(s)}_{\langle g\rangle}\right)_j g^{H_1'(\texttt{ID})} = g^{s_j+H_1'(\texttt{ID})}$.
2. P_i performs $\left(\mathcal{C}^{(w,w')}_{\langle\hat{g},\hat{h}\rangle}, w_i, w_i'\right) = \mathsf{Mul}_{\mathsf{Ped}}(n,t,\hat{g},\hat{h},\left(\mathcal{C}^{(s^{\texttt{ID}})}_{\langle g\rangle}, s_i^{\texttt{ID}}\right), \left(\mathcal{C}^{(z,z')}_{\langle\hat{g},\hat{h}\rangle}, z_i, z_i'\right))$, where $w = s^{\texttt{ID}}z$ $= (s+H_1'(\texttt{ID}))z$ and $w' = (s+H_1'(\texttt{ID}))z'$ and sends $\left(\mathcal{C}^{(w)}_{\langle\hat{g},\hat{h}\rangle}, w_i\right)$ along with NIZKPK$_{\equiv Com}(w_i, w_i'$, $\left(\mathcal{C}^{(w)}_{\langle\hat{g}\rangle}\right)_i, \left(\mathcal{C}^{(w,w')}_{\langle\hat{g},\hat{h}\rangle}\right)_i)$ 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 $\left(\mathcal{C}^{(z)}_{\langle\hat{g}\rangle}\right)_i = \hat{g}^{z_i}$ along with NIZKPK$_{\equiv Com}(z_i, z_i', \left(\mathcal{C}^{(z)}_{\langle\hat{g}\rangle}\right)_i, \left(\mathcal{C}^{(z,z')}_{\langle\hat{g},\hat{h}\rangle}\right)_i)$ to the client.
4. The client verifies $\left(\mathcal{C}^{(z)}_{\langle\hat{g}\rangle}\right)_i$ using the received NIZKPK$_{\equiv Com}$, 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 [20] define an IND-ID-CCA secure version of the SK-IBE scheme secure against the BDHI assumption. Here, the random oracle H_1 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 [27] to achieve IND-ID-CCA security.
Encryption: To encrypt a message M of some fixed bit length ℓ for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^\ell$, 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 5.2. *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 is 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 5.1 for (n,t)-$\mathsf{FullIdent}$. To prove Theorem 5.2 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 [20, §3.2]. Note that $\mathsf{SKBasicPub}^{hy}$ is not a distributed scheme. Therefore, recalling Lemma 2 and 3 from [20] to prove $\mathsf{SKBasicPub}^{hy} \to \mathsf{SKBasicPub}$ and $\mathsf{SKBasicPub} \to \mathsf{BDHI}$ respectively we complete the proof of Theorem 5.2. Next, we prove (n,t)-SK-IBE $\to$ $\mathsf{SKBasicPub}^{hy}$.

Lemma 5.4. *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 [20, §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 5.3. 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 oracle H_1' and H for $\mathcal{A}_1$, where H is a random oracle required in $\text{NIZKPK}_{\equiv Com}$.

H'_1 **queries:** Same as in [20, §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, 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 NIZKPK$_{\equiv Com}$ 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 $\pi_{\equiv Com} = (\tau, u_1, u_2)$ 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 NIZKPK$_{\equiv Com}$, 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 [20, §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. □

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

BB$_1$-IBE belongs to the commutative-blinding IBE family. Boneh and Boyen [9] proposed the original scheme with a security reduction to the decisional BDH assumption [37] in the standard model against selective-identity attacks. However, with a practical requirement of security against adaptive-identity chosen-ciphertext attacks (IND-ID-CCA), in the recent IBCS standard [13], 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 [12], 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 [12] and [13].

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 [12], Boyen does not include the parameters $\hat{g}$ and $\hat{g}^\beta$ from the original BB$_1$ scheme [9] 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 §5.4, this extra generator $\hat{h} \in \hat{\mathbb{G}}$ is precomputed using the $\mathsf{Random}_{\mathsf{Feld}}$ 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 $\mathsf{Random}_{\mathsf{Feld}}$ protocol invocations:

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

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 $\alpha_i\beta_i + (\alpha_i H'_1(\texttt{ID}) + \gamma_i)r_i$ and provide the corresponding $\hat{g}^{\alpha_i\beta_i + (\alpha_i H'_1(\texttt{ID}) + \gamma_i)r_i}, \hat{g}^{r_i}$ to the client, who now needs $2t + 1$ valid shares to obtain her private key. However, $\alpha_i\beta_i + (\alpha_i H'_1(\texttt{ID}) + \gamma_i)r_i$ 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 privacy 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 $w_i = (\alpha\beta + (\alpha H'_1(\texttt{ID}) + \gamma)r)_i$ is a share of random degree t polynomial. The protocol works as follows.

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

2. P_i computes its share w_i of $w = \alpha\beta + (\alpha H'_1(\texttt{ID}) + \gamma)r$ using protocol $\mathsf{Mul}_{\mathsf{BP}}$ in Eq. 11.
$$\left(\mathcal{C}^{(w)}_{\langle g^* \rangle}, w_i\right) = \mathsf{Mul}_{\mathsf{BP}}(n, t, f, g^*, desc, \left(\mathcal{C}^{(\alpha)}_{\langle g \rangle}, \alpha_i\right), \left(\mathcal{C}^{(\beta)}_{\langle \hat{h} \rangle}, \beta_i\right), \left(\mathcal{C}^{(\gamma)}_{\langle g \rangle}, \gamma_i\right), \left(\mathcal{C}^{(r)}_{\langle \hat{h} \rangle}, r_i\right))$$
where $desc = \{(1, 1, 2), (H'_1(\texttt{ID}), 1, 4), (1, 3, 4)\}$ is the description of the required binary product under the ordering $(\alpha, \beta, \gamma, r)$ of secrets. To justify our choices of commitment generators, we present the pairing-based verification in protocol $\mathsf{Mul}_{\mathsf{BP}}$:
$$e(g^{\alpha_i\beta_i + (\alpha_i H'_1(\texttt{ID}) + \gamma_i)r_i}, \hat{h}) \stackrel{?}{=} e(g^{\alpha_i}, \hat{h}^{\beta_i}) e((g^{\alpha_i})^{H'_1(\texttt{ID})} g^{\gamma_i}, \hat{h}^{r_i})$$
.
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 $g^{\alpha\beta + (\alpha H'_1(\texttt{ID}) + \gamma)r}$.

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 Lagrange-interpolates the valid received shares to generate her private key $(\hat{g}^{\alpha\beta + (\alpha H'_1(\texttt{ID}) + \gamma)r}, \hat{g}^r)$. 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 §5.4, online execution of the $\mathsf{Random}_{\mathsf{Feld}}$ 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 §5.5, we use the BB$_1$-IBE version defined in [12] and [13] for the encryption and decryption protocols here. Boyen [12] claims IND-ID-CCA security of this system against the BDH assumption. This scheme uses $H'_3 = G_t \times \{0,1\}^\ell \times \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$ along with H'_1 and H_2 from SK-IBE.
Encryption: To encrypt a message M of some fixed bit length ℓ for a receiver of identity $\mathtt{ID}$, a sender chooses $\sigma \in_R \{0,1\}^\ell$, computes $k = (e(g,\hat{g})^{\alpha\beta})^\sigma$ and $h_{\mathtt{ID}} = H_1(\mathtt{ID})$, 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 BB$_1$-IBE 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 5.3. *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 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 $\mathcal{A}_3$. 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 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 $\mu_3 a + \nu_3 = -\alpha h^*_{\mathtt{ID}} + \alpha'$, where $h^*_{\mathtt{ID}} = -\mu_3/\mu_1$ is a challenge identity-hash and $\alpha' = \nu_3 - \nu_1\mu_3/\mu_1 = \alpha h^*_{\mathtt{ID}} + \gamma$. 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'^{list}_1, $\mathcal{B}$ chooses $j \in_R \{1, \ldots, q_{H_1}\}$. When $\mathcal{A}$ queries H'_1 for $\mathtt{ID}_i$, $\mathcal{B}$ proceeds as follows: if $i \neq j$, it picks $h_{\mathtt{ID}_i} \in_R \mathbb{Z}_p$, adds a tuple $\langle \mathtt{ID}_i, h_{\mathtt{ID}_i} \rangle$ and gives back $h_{\mathtt{ID}_i}$ to $\mathcal{A}$. If $i = j$, it sets $\langle \mathtt{ID}_j, h^*_{\mathtt{ID}} \rangle$. Note that multiple queries for the same identity are answered with the corresponding entry in its H'^{list}_1. Further, the output of H'_1 is uniformly distributed in $\mathbb{Z}_p$ and independent of 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 presents, $\mathcal{B}$ responds with the associated response, else $\mathcal{B}$ sends a random element of the appropriate size as its response, adds an input and response tuple in the oracle list. The corresponding (random) oracle list entries look as follows: $H^{list}_2(\mathbb{G}_t, \{0,1\}^\ell) = \langle k_i, h_{k_i} \rangle$, and $H'^{list}_3(\mathbb{G}_t, \{0,1\}^\ell, \mathbb{G}, \mathbb{G}) = \langle k_i, \rho_i, \rho_{0i}, \rho_{1i} \rangle$, where ℓ is the message length.
Phase 1 **- Extraction Queries:** When $\mathcal{A}$ asks for the private key for $\mathtt{ID}_i$, $\mathcal{B}$ first gets $H'_1(\mathtt{ID}_i) = h_{\mathtt{ID}_i}$. 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 $\Delta h = h_{\mathtt{ID}_i} - h^*_{\mathtt{ID}}$.

Using $\hat{h}^{\tilde{r}}$ and 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 NIZKPK$_{\equiv Com}$ for each entry in $\mathcal{C}^{(\tilde{r})}_{\langle \hat{h} \rangle}$, $\mathcal{A}_2$ 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 $\pi_{\equiv Com} = (\tau_i, u_{1i}, u_{2i})$ to $\mathcal{A}$.

It then computes $d'_0 = (g^*)^{-\beta\alpha'/\Delta h}(g^*)^{\alpha r \Delta h + \alpha' r} = (g^*)^{\alpha\beta + (\alpha h_{\mathtt{ID}_i} + \gamma)\tilde{r}}$ and using known shares of α_i, β_i and γ_i for $P_i \in P_{Bad}$, it runs $\mathsf{Mul}_{\mathbf{BP}}$ for $w = \alpha\beta + (\alpha h_{\mathtt{ID}_i} + \gamma)\tilde{r}$. 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 $(\mathtt{ID}_i, C_i)$ as follows. $\mathcal{B}$ first gets $H'_1(\mathtt{ID}_i) = h_{\mathtt{ID}_i}$. If $i \neq j$, $\mathcal{B}$ obtains the private key $(\hat{g}^{\alpha\beta + (\alpha h_{\mathtt{ID}_i} + \gamma)\tilde{r}}, \hat{g}^{r})$ 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 $H_3'^{list}$ 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 $H_3'^{list}$. 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, 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 $(\rho_b \in_R \{0,1\}^{\ell}, \rho_{0b} = g^c, \rho_{1b} = (g^c)^{\alpha'}, t_b \in_R \mathbb{Z}_p)$ 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 correct D with probability $\epsilon(\kappa)/(q_{H'_1} q_{H'_3})$. □

Note that 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 [33]. Further, we do not discuss the liveness and consistency properties for our asynchronous protocols as liveness and consistency of all the distributed primitives provides liveness and consistency for the distributed PKG setup and distributed key extraction protocols.

6 System Aspects

In this section, we discuss the system aspects of distributed PKGs. As DKG is by far the most important component of our distributed PKGs, we first implement and test the DKG protocol [40] that we use in our distributed PKGs. In the process, we propose several system-level optimizations for this DKG. We also analyze practical aspects of our distributed PKGs and present a comparative study. Finally, we mention proactive security and group modification protocols for our distributed PKGs.

Note that two distributed CAs for PKC, Ω [53] and Cornell Online Certification Authority (COCA) [61], have been designed previously. However, with their focus on CAs, the protocols they provide are mis-

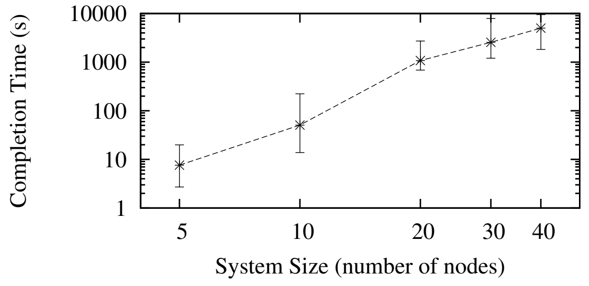

Figure 1: Completion Time (with min/max bars) vs System Size (log-log plot)

matched to the requirements of a distributed PKG. As a result, we do not design our distributed PKGs using these solutions.

6.1 DKG Implementation on PlanetLab

We design our DKG nodes as state machines (using the state machine replication approach [43, 55]), where nodes move from one state to another based on messages received. Messages are categorized into three types: operator messages, network messages and timer messages. The operator messages define interactions between nodes and their operators, the network messages realize protocol flows between nodes, and the timer messages implement the weak synchrony assumption described in $\S 3.1$.

We aim at building a distributed PKG for IBE schemes. Therefore, we develop our object-oriented C++ implementation over the PBC library [45] for the underlying elliptic-curve and finite-field operations and a PKI infrastructure with DSA signatures based on GnuTLS [46] for confidentiality and message authentication. (Note that *nodes* have TLS PKI certificates, which does not conflict with the goal of providing IBE private keys to *clients*.) In order to examine its realistic performance, we test our DKG implementation on the PlanetLab platform [52].

Performance Analysis. We test the performance of our DKG implementation for systems of up to 40 nodes and we observe an expected approximately cubic growth in the average completion time.[2] Figure 1 presents our results in graphical form. In practical applications such as [41], these values, ranging from seconds to a little over an hour, are small as compared to DKG phase sizes (in days). Importantly, the use of dedicated high-performance servers instead of unreliable resource-shared PlanetLab nodes can drastically improve the performance. We also measure minimum and maximum completion times for the experiments. Big 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.

To check the applicability of the weak synchrony assumption [18] that we use in DKG, 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 $delay(t)$ functions and we suggest that this should only be finalized for a system after rigorous testing.

While implementing this system, we also found two system-level optimizations for this DKG.

- To the original DKG protocol, we add a new shared network message from a node to a leader having $2t+f+1$ signed ready messages for a completed VSS. The leader can then include this VSS instance in its DKG send without completion of the VSS instance at its own machine.

[2]With cubic message complexity, larger distributed systems ($n > 50$) are not practical for the Internet.

Table 1: Operation count and key sizes for distributed PKG setups and distributed private-key extractions (per key)

		BF-IBE		**SK-IBE**		**BB$_1$-IBE**	
		Setup	Extraction	Setup	Extraction	Setup	Extraction
Operation Count:	Generator h or $\hat{h}$	X		$\surd$		$\surd$	
	DKG-Sh[a]						
	(precomputed)	-	0	-	1^P	-	1^P
	(online)	1^F	0	1^F	1^P	3^F	1^F
	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 DKG-Sh F indicates use of Feldman 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.

- During our experiments, we observed that the VSS instances are more resource consuming than the agreement required at the end. Except during the Mul protocol, we only need $t+1$ VSS instances to succeed. Assuming $t+f$ VSS instances might fail during a DKG, it is sufficient to start VSSs 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 to when to start a VSS instance if required.

6.2 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 §5. For a detailed comparison of the encryption and decryption algorithms of BF-IBE, SK-IBE and BB$_1$-IBE, we refer readers to the survey by Boyen [12]. 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 1 provides a detailed operation count and key size comparison of our three distributed PKGs. We count DKG-Sh instances, pairings, NIZKPKs, interpolations and public and private key sizes. We leave aside the comparatively small exponentiations and other group operations. As mentioned in §5.5, for BB$_1$-IBE, with curves 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 §5.3, 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 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 suggest 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 suggest 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 H_1 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.

6.3 Proactive Security and Group Modification

With an endless supply of software and network security flaws, system attacks not only are prevalent but have also been growing. The distributed nature of our protocols mitigates the effects of those attacks to some extent, but their time-independence makes them vulnerable to a *gradual break-in* by a *mobile attacker* breaking into system nodes one by one. The concept of proactive security [49] has been introduced to counter these attacks. Further, on a long-term basis, the set of PKG nodes will need to be modified, which can also cause changes to the system's security threshold t and the crash-limit f. Therefore, for our distributed PKG systems, we need proactive security and group modification protocols.

We observe that the proactive security and group modification protocols defined in [40], 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 [9] or Waters' IBE [60].

7 Conclusion

In this paper, 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. 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. Further, 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.

While developing our distributed PKGs, we also developed asynchronous computational protocols for distributed multiplication and distributed inverse computation, which may have their own applications. To confirm the feasibility of a distributed PKG in the asynchronous communication model, we also implemented and verified the efficiency and the reliability of an asynchronous DKG protocol using extensive testing over the PlanetLab platform. We also suggested proactive security and group modification protocols for our distributed PKGs. In the future, we would like add those features to our implementation.

Acknowledgements

This work is supported by NSERC, MITACS, and a David R. Cheriton Graduate Scholarship. We thank Sanjit Chatterjee, Alfred Menezes and Kenny Paterson for helpful discussions and suggestions.

References

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

[2] S. S. Al-Riyami and K. G. Paterson. Certificateless Public Key Cryptography. In *ASIACRYPT'03*, pages 452–473, 2003.

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

[4] J. Bar-Ilan and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *PODC'89*, pages 201–209, 1989.

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

[6] 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.

[7] G. R. Blakley. Safeguarding Cryptographic Keys. In *the National Computer Conference*, pages 313–317, 1979.

[8] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography (PKC'03)*, pages 31–46, 2003.

[9] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In *EUROCRYPT'04*, pages 223–238, 2004.

[10] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO'01*, pages 213–229, 2001.

[11] X. Boyen. General *Ad Hoc* Encryption from Exponent Inversion IBE. In *EUROCRYPT'07*, pages 394–411, 2007.

[12] X. Boyen. A Tapestry of Identity-based Encryption: Practical Frameworks Compared. *IJACT*, 1(1):3–21, 2008.

[13] 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.

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

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

[16] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *CRYPTO'99*, pages 98–115, 1999.

[17] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC'93*, pages 42–51, 1993.

[18] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[19] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO'92*, pages 89–105, 1992.

[20] L. Chen and Z. Cheng. Security Proof of Sakai-Kasahara's Identity-Based Encryption Scheme. In *IMA Int. Conf.*, pages 442–459, 2005.

[21] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *FOCS'85*, pages 383–395, 1985.

[22] X. Chunxiang, Z. Junhui, and Q. Zhiguang. A Note on Secure Key Issuing in ID-based Cryptography. Technical report, 2005. `http://eprint.iacr.org/2005/180`.

[23] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of ACM*, 35(2):288–323, 1988.

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

[25] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*, pages 186–194, 1986.

[26] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, 1985.

[27] E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO'99*, pages 537–554, 1999.

[28] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[29] D. Galindo. Boneh-Franklin Identity Based Encryption Revisited. In *ICALP'99*, pages 791–802, 2005.

[30] R. Gangishetti, M. C. Gorantla, M. Das, and A. Saxena. Threshold key issuing in identity-based cryptosystems. *Computer Standards & Interfaces*, 29(2):260–264, 2007.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *EUROCRYPT'99*, pages 295–310, 1999.

[32] 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.

[33] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[34] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In *PODC'98*, pages 101–111, 1998.

[35] V. Goyal. Reducing Trust in the PKG in Identity Based Cryptosystems. In *CRYPTO'07*, pages 430–447, 2007.

[36] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *ANTS-IV*, pages 385–394, 2000.

[37] A. Joux. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *ANTS-V*, pages 20–32, 2002.

[38] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, 2003.

[39] M. Joye and G. Neven. *Identity-Based Cryptography - Volume 2 Cryptology and Information Security Series*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008.

[40] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS'09)*, pages 119–128, 2009.

[41] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *7th Privacy Enhancing Technologies Symposium (PET'07)*, pages 95–112, 2007.

[42] 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.

[43] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.

[44] 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.

[45] B. Lynn. PBC Library – The Pairing-Based Cryptography Library. `http://crypto.stanford.edu/pbc/`, 2009. Accessed April 2009.

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

[47] A. Menezes, P. V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1997.

[48] 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.

[49] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks (Extended Abstract). In *PODC'91*, pages 51–59, 1991.

[50] T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Eurocrypt'91*, pages 522–526. Springer-Verlag, 1991.

[51] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, pages 129–140, 1991.

[52] 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.

[53] M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright. The Omega Key Management Service. *Journal of Computer Security*, 4(4):267–288, 1996.

[54] R. Sakai and M. Kasahara. ID based Cryptosystems with Pairing on Elliptic Curve. Cryptology ePrint Archive, Report 2003/054, 2003.

[55] F. B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

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

[57] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[58] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *CRYPTO'84*, pages 47–53, 1984.

[59] H. Wang, Y. Zhang, and D. Feng. Short Threshold Signature Schemes Without Random Oracles. In *INDOCRYPT'03*, pages 297–310, 2005.

[60] B. Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT'05*, pages 114–127, 2005.

[61] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed online certification authority. *ACM Trans. Comput. Syst.*, 20(4):329–368, 2002.

[62] 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.

A Non-interactive Zero-knowledge Proofs

We now present the details of the non-interactive zero-knowledge proofs of knowledge (NIZKPKs) introduced in §4.2. Here, H is a hash function modelled by a random oracle.

The first proof is that a Feldman commitment $F = \mathcal{C}_{\langle g \rangle}(s) = g^s$ and a Pedersen commitment $P = \mathcal{C}_{\langle g,h \rangle}(s, r) = g^s h^r$ are both committing to the same value s. We denote this by NIZKPK$_{\equiv Com}(s, r, F, P)$. The proof is equivalent to zero-knowledge proofs of knowledge used by Canetti *et al.* in their adaptive secure DKG [16].

The proof is generated as follows:

- Pick $v_1, v_2 \in_R Z_p$
- Let $t_1 = g^{v_1}$, $t_2 = h^{v_2}$
- Let $\tau = H(g, h, F, P, t_1, t_2)$
- Let $u_1 = v_1 - \tau \cdot s \pmod{p}$, $u_2 = v_2 - \tau \cdot r \pmod{p}$
- The proof is $\pi_{\equiv Com} = (\tau, u_1, u_2)$

The verifier checks this proof (given $\pi_{\equiv Com}$, g, h, F, P) as follows:

- Let $t'_1 = g^{u_1} F^{\tau}$, $t'_2 = h^{u_2}(P/F)^{\tau}$
- Accept the proof as valid if $\tau = H(g, h, F, P, t'_1, t'_2)$

The second proof is that two Feldman commitments $F_1 = \mathcal{C}_{\langle g \rangle}(s) = g^s$ and $F_2 = \mathcal{C}_{\langle h \rangle}(s) = h^s$ commit to the same value; that is, the discrete logs of F_1 and F_2 to the bases of g and h respectively are equal. We denote this by NIZKPK$_{\equiv DLog}(s, F_1, F_2)$. The proof is standard [19]:

The proof is generated as follows:

- Pick $v \in_R Z_p$
- Let $t_1 = g^v$, $t_2 = h^v$
- Let $\tau = H(g, h, F_1, F_2, t_1, t_2)$
- Let $u = v - \tau \cdot s \pmod{p}$
- The proof is $\pi_{\equiv DLog} = (\tau, u)$

The verifier checks this proof (given $\pi_{\equiv DLog}$, g, h, F_1, F_2) as follows:

- Let $t'_1 = g^u F_1^{\tau}$, $t'_2 = h^u F_2^{\tau}$
- Accept the proof as valid if $\tau = H(g, h, F_1, F_2, t'_1, t'_2)$