A Fully Resilient, Identity-based, Efficient, Non-interactive and Decentralized Key Exchange Protocol (FRIEND-KEP)

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Amit Singh Bhati

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2019

Supervisor: Sanjit Chatterjee

Certificate

This is to certify that this dissertation entitled *"A Fully Resilient, Identity-based, Efficient, Non-interactive and Decentralized Key Exchange Protocol"* towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents the study and work carried out by Amit Singh Bhati at the Department of Computer Science and Automation, Indian Institute of Science (IISc) under the supervision of Dr. Sanjit Chatterjee, during the academic year 2018-19.

[illegible]

Dr. Sanjit Chatterjee

Committee:

Dr. Sanjit Chatterjee

Dr. Deepak Dhar

I would like to dedicate this thesis
to my loving family,
friends and teachers.

Declaration

I hereby declare that the matter embodied in the report entitled *"A Fully Resilient, Identity-based, Efficient, Non-interactive and Decentralized Key Exchange Protocol"*, are the bonafide results of the work carried out by me at the department of Computer Science and Automation, Indian Institute of Science (IISc) under the supervision of professor Sanjit Chatterjee, and the same has not been submitted elsewhere for any other degree.

Amit

Amit Singh Bhati
March 2019

Acknowledgements

First and foremost, I would like to extend my profound gratitude to my thesis supervisor, Dr. Sanjit Chatterjee for providing me the opportunity to carry out my MS thesis project under his supervision. I would like to thank him for his invaluable guidance and support throughout my thesis project. He has made me recognize my fullest potential and encouraged me to achieve the highest level of professionalism. I am also thankful to my TAC member, Dr. Deepak Dhar for providing motivation to me throughout the course of my project.

Many thanks to all of my IISER friends especially Nishad, Ojha, Amol, Suhel, Horo, Ajinkya, Rahul, Kaushik, Nishant, Niranjana, Sanjay, Chinmay, Saurath, Kirtikesh, Tilva, Pavan, Dharmendra and Shubham for making my last five years enjoyable and exciting. Out of IISER, I would like to thank my close friend and former batchmate Amey for keeping in touch and motivating me.

I am grateful to my labmates at Informatics and Security Lab, IISc - Mayank, Akash, Shravan, Kabaleesh, Tapas, Sayantan and Sonali for their constant support and all the helpful discussions throughout my project. It was truly a pleasure working alongside them.

Last but not the least, my heartfelt thanks to all my family members and my girlfriend Ranjana, for their unconditional love and support. I owe my deepest gratitude to each one of them.

Abstract

A non-interactive key exchange (NIKE) allows two parties to compute a unique shared key without any interaction. Since the innovative work of Diffie and Hellman [10], NIKE has become one of the fundamental problems of modern cryptography. Identity-based NIKE (ID-NIKE) is a fundamental primitive of Identity Based Cryptography. It allows a party to compute a shared key using its own secret key and the other party's identity. In the recent past, where identity-based encryption and signature have been thoroughly explored, ID-NIKE didn't get enough attention. At the moment, we have only few ID-NIKE protocols available (with no *fully* secure Hierarchical-ID-NIKE (H-ID-NIKE) protocol) in the literature.

Mobile Ad-hoc Networks (MANETs) are decentralized networks of mobile devices with limited resources in terms of storage, power, computation, communication etc. They encounter some serious security issues due to their high mobility and hierarchical structure. H-ID-NIKE can be used to establish shared secret keys in MANETs using minimal resources. Secure H-ID-NIKE protocols are highly appreciated for security sensitive applications in MANETs such as in military or tactical networks.

In particular, key exchange protocols with the four functional properties (as posed by Gennaro *et al.* in [16]) are considered well-suited for the MANET environment i.e., the protocol should be *non-interactive, identity-based, hierarchical* and *fully resilient* against arbitrary number of node compromises at any level. However, the proposed solution for this problem in [16] does not really satisfy all four properties. Their protocol is neither fully resilient nor allows a secure key exchange at any non-leaf level. Later in 2017, Tiwari proposed another H-ID-NIKE construction (named BIOS-SOK) [27] as a possible solution for this problem. BIOS-SOK is a non-interactive, identity-based and hierarchical key exchange protocol which allows multi-level shared key computations. However, it is shown secure under a restricted security model and is not fully resilient in practical scenarios. There are few more constructions available in the literature ([24, 11, 5, 23, 20, 25]) which contain three of these four properties. However, there is no fully secure and practical key exchange protocol with all four properties.

In this thesis, we have proposed a key exchange protocol (named α-BSOK) as a possible solution for this open problem. α-BSOK is non-interactive, identity-based, hierarchical, efficient and fully resilient against arbitrary number of node corruptions. α-BSOK is based on the idea of BIOS-SOK hybrid[27]. It is a hybrid of two non-hierarchical protocols BIOS[20] and SOK[25]. Both of these protocols are non-interactive, identity-based and fully resilient. In our work, we have made a hybrid of these protocols which is hierarchical in nature. We have provided a rigorous security analysis for α-BSOK in a stronger security model (compared to [12, 27, 16]). We have discussed two variants of the α-BSOK protocol (named as β-BSOK and β-BSOK-KWT) that slightly traded efficiency for better security. We have also done an implementation and simulation data analysis of α-BSOK with other existing protocols to compare its efficiency.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

Key exchange is a fundamental tool for secure communication. It is used to compute a common secret between two parties over an untrusted channel. Such a common secret can be used by the parties as a key for their secure communication. Traditional ways of key exchange used to involve some secure physical channels, such as key tapes or printed key lists which were transported by trusted couriers and diplomatic bags with certain legal protections. This process used to require a lot of money and manpower which was not always practical.

In 1976, Diffie and Hellman introduced the first secure and practical key exchange protocol (known as the DH protocol) in their ground-breaking paper “New Directions in Cryptography” [10]. The DH protocol allows two parties to agree on a common secret key over a public channel. This protocol is also known as the first Non-Interactive Key Exchange (NIKE) protocol in the public key setting. NIKE allows two parties to compute a common secret key without any interaction. In DH protocol, Each party maintains two different keys known as a public and a secret key. The public key of a party is publicly available and is used by the other party to initiate a key exchange. Each party is also required to exchange a public key certificate with a Certification Authority (CA). A public key certificate is required to confirm the authenticity of the corresponding party’s public key. However, management of these public key certificates involves a complex Public Key Infrastructure (PKI). Certificate management cost is one of the main disadvantages of public key cryptography.

Later, in 1984, Shamir proposed the idea of Identity-based Cryptography (IBC) to avoid the problem of expensive certificate management [26]. IBC is a type of public-key cryptography which uses publicly known strings such as phone number, email or IP address as a public key for the parties in the system. Thus, the authenticity of a public key can be easily verified from the publicly available information. In IBC, a trusted third party, called Private Key Generator (PKG) generates and distributes the secret keys of the other parties.

The first identity-based NIKE (ID-NIKE) protocol (known as the SOK protocol) was introduced by Sakai, Ohgishi and Kasahara in 2000 [25]. SOK is a non-interactive protocol which uses bilinear pairing to compute shared secret keys between two parties. SOK was followed by a few more ID-NIKE protocols including Trapdoor Discrete Logarithm based protocol (TDL)[23] and Basic ID-based One-way-function Scheme (BIOS)[20] which do not use pairing for the key exchange. However, all of these ID-NIKE protocols are only applicable to a flat architecture network i.e., only PKG can generate and distribute the secret keys in the network. On the other hand, in hierarchical networks such as military organizations we expect a decentralized key distribution i.e., every node should be able to generate and distribute secret keys to any of its descendants. Thus, these ID-NIKE protocols are not directly applicable to such applications. An ID-NIKE protocol is said to be hierarchical if it allows the decentralized key distribution.

In networks like MANETs or military networks, nodes are mobile and are at a high risk of being compromised. When a node is compromised, all the secret keys of that node are revealed to the attacker. It is possible for a key exchange protocol that once the number of compromised nodes grows above some threshold, an attacker can learn keys of uncompromised nodes. We call a protocol to be fully resilient, if the communication between any two uncompromised nodes remain secure even after arbitrary number of node compromises. We expect an ideal ID-NIKE to be fully resilient.

The first hierarchical ID-NIKE (H-ID-NIKE) protocol (named as HHKAS) was proposed by Gennaro *et al.* in 2008 [16]. Though HHKAS supports a proper hierarchical key distribution, it only allows the nodes at leaf level to do a proper key-exchange. For non-leaf level nodes, the key exchange is neither guaranteed nor shown secure. Also HHKAS is fully resilient only at the leaf level and for the non-leaf levels it is resilient against node compromises up to some defined thresholds. Later, in 2014, Freire proposed another construction[12] for H-ID-NIKE which can be treated as a generalized version of SOK protocol. Freire's protocol uses multilinear maps to compute shared secret keys between two parties. However, there are serious security issues with multilinear maps [8] and thus this protocol is not practical for real-life applications. In 2017, Tiwari proposed BIOS-SOK hybrid as a possible solution for the H-ID-NIKE problem. BIOS-SOK supports hierarchical key distribution and allows multi-level shared key computations. However, it is shown secure under a restricted security model and is not fully resilient in practical scenarios.

Our goal in this thesis is to propose a protocol that has all the above functional properties and is secure in a quite stronger sense i.e., it should be *non-interactive* (to reduce communication and power costs), *identity-based* (to avoid the costs required for PKI management), *hierarchical* (to allow decentralized key distribution) and *fully resilient* against arbitrary number of node compromises at any level in the hierarchy.

In this thesis, we have proposed a new protocol (along with its two variants) based on the idea of BIOS-SOK. Our proposed protocol contains all of these four properties under a stronger security model. We have also presented a comparative analysis of our proposed solution with other existing protocols.

1.2 Plan of the Thesis

Chapter 2 contains the preliminary topics and necessary definitions which are required to understand the work presented in this thesis. In Chapter 3, we have reported a literature survey about H-ID-NIKE protocols including their security models and some limitations. This survey mainly covers two existing H-ID-NIKE protocols named as HHKAS[16] and BIOS-SOK[27]. These protocols claim to possess the four functional properties as posed by Gennaro *et al.* in [16]. However, in Chapter 3, we have explained that the HHKAS and BIOS-SOK protocol satisfy a quite restricted definition of H-ID-NIKE and their underlying security models are weaker than a general hierarchical security model named as H-IND-SK model (see Section 3.2).

Chapters 4 and 5 are the main contribution of this thesis. In Chapter 4, we have proposed a new hierarchical key exchange protocol using two existing non-hierarchical protocols - BIOS[20] and SOK[25]. We claim that our proposed protocol contains all four functional properties as mentioned in [16] and is also secure in the H-IND-SK model. We have also presented two additional variants of this protocol that slightly traded efficiency for better security.

The prototype implementation and simulation part of our work is presented in the Chapter 5. Here, we have provided an efficiency comparison between our designed protocol and HHKAS in terms of time and storage requirements. We have also presented a simulation study on these two protocols using the network simulator NS2[17] under a MANET model. In the end, we have summarized our work and the study with an analytic comparison of relevant protocols over different types of essential properties. Based on all these comparison, we have concluded that our designed protocol (named as α-BSOK) is a better choice over the existing protocols for most of the real-life applications. We have also mentioned some of the remaining limitations of α-BSOK.

In Chapter 6, we have concluded this thesis with some open problems and future directions for research in this area and related fields.

Chapter 2

Preliminary Topics

2.1 Mobile Ad-hoc Networks

Mobile Ad-hoc Networks (MANETs) are decentralized, wireless networks of mobile and resource-constrained devices. Usually these constraints are in terms of storage, power, computation and communication. Laptops, smart phones, unmanned aerial vehicles and military MANETs are some examples of MANET devices. In recent years, there is a huge increase in the usage of MANETs for unmanned army system. They are very useful in both surveillance and future combat systems. However, MANETs encounter some real security issues due to their high mobility, dynamic topology and hierarchical structure. Hierarchical Identity-based Non-interactive Key Exchange (H-ID-NIKE) can be an interesting solution for secure communication between two nodes in MANETs. Secure H-ID-NIKE protocols are highly appreciated for security sensitive applications such as in military or tactical networks. For further details on MANETs, we point the reader to [28].

2.2 Mathematical Background

Let us denote the set of natural numbers by $\mathbb{N}$, the set of integers by $\mathbb{Z}$ and the set of integers modulo n by $\mathbb{Z}_n$. We define $\mathbb{Z}_n^* \subseteq \mathbb{Z}_n$ as a set containing elements relatively prime to n. It is easy to see that if n is a prime, then $\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\}$. We hereby provide some basic definitions which we have used in this thesis.

- **Polynomial-time algorithm:** A polynomial-time algorithm is an algorithm whose running time can be bounded by a polynomial on the size of the input i.e., for some given input size n and a constant k, the worst-case running time for such algorithm is of the form $\mathcal{O}(n^k)$. All polynomial time algorithms are considered to be "efficient" algorithms.

- **Negligible function:** A function $\epsilon(n)$ is said to be negligible if for all $c \in \mathbb{N}$ there exists an integer $N_0 \in \mathbb{N}$ such that for every $n > N_0$, we have -

$$\epsilon(n) < \frac{1}{n^c} .$$

  In other words, we can say that a function is negligible if it grows slower than inverse of all possible positive polynomials.

- **Security parameter:** In cryptography, a security parameter λ (generally expressed as a unary string 1^λ) is a measure which explains how "hard" it is for an adversary to break the underlying cryptographic protocol. Informally, the security parameter measures the level of security in which a cryptographic protocol is deployed. For example, if a protocol is deployed in λ-bit security level then an adversary is required to do $\mathcal{O}(2^\lambda)$ number of operations to break the protocol.

- **Bilinear pairing:** Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be three cyclic groups of prime order q. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is said to be an admissible pairing if the following conditions are satisfied:

  1. **Bilinearity:** for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_q$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
  2. **Non-Degeneracy:** For all pairs (g_1, g_2) with $\langle g_1 \rangle = \mathbb{G}_1$, $\langle g_2 \rangle = \mathbb{G}_2$, we have $\langle e(g_1, g_2) \rangle = \mathbb{G}_T$.
  3. **Efficiently Computable:** There exists an efficient algorithm which can compute $e(g_1, g_2)$, for all pairs $(g_1, g_2) \in \mathbb{G}_1 \times \mathbb{G}_2$.

  When $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is said to be *symmetric*.

- **Elliptic Curve (over finite field):** Let $\mathbb{F}_q$ be a finite field of characteristic $p > 3$. An elliptic curve E over $\mathbb{F}_q$ is a plane algebraic curve defined by an equation of the form $E : y^2 = x^3 + ax + b$ where $a, b \in \mathbb{F}_q$ and $4a^3 + 27b^2 \neq 0$. The curve E can be considered as a set of points (x, y) satisfying the equation $y^2 = x^3 + ax + b$ with the special point $\mathcal{O}$ (the point at infinity). In other words, if the set of points on E is given by $E(\mathbb{F}_q)$ then $E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$.

For further details on elliptic curves and pairings, we direct the reader to [14, 15]. Elliptic curves are some of the main structures which have been used in practice to construct efficient and cryptographically secure pairings. Weil and Tate pairings are examples of bilinear pairings which are defined using elliptic curve groups over finite fields [30, 13, 6].

Pairings have been used in Cryptography to design innovative protocols for various purposes including key exchange. The first known bilinear pairing based key exchange protocol named as the three-party one-round NIKE protocol was proposed by Antonie Joux in 2000 [18]. Later, in the same year, Sakai, Ohgishi and Kasahara [25] proposed the first identity-based NIKE protocol using bilinear pairing.

Symmetric pairings usually make protocol description and their security proofs simpler to understand. For further work and discussion in this thesis, we will consider a bilinear pairing to be a symmetric pairing defined as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\mathbb{G}_T$ is a multiplicative group of prime order q and $\mathbb{G}$ is an elliptic curve group over $\mathbb{F}_q$.

2.3 Hardness Assumptions

Security of the protocols described in this thesis rely on the hardness of the Computational Bilinear Diffie-Hellman (CBDH) problem (introduced in [6]) or one of its variants. In this section, we define the problems and their corresponding assumptions which are relevant to our work.

1. **Computational Bilinear Diffie-Hellman Assumption (CBDH)**
   Given a randomly chosen $P \in \mathbb{G}$, as well as aP, bP, cP for unknown, randomly chosen $a, b, c \in \mathbb{Z}_q^*$, it is hard to compute $e(P,P)^{abc}$. Formally, an algorithm $\mathcal{A}$ has advantage ϵ in solving the CBDH in $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$ if

   $$Pr[\mathcal{A}(P, aP, bP, cP) = e(P,P)^{abc}] \geq \epsilon$$

   where the probability is over the random choice of $a, b, c \in \mathbb{Z}_q^*$, $P \in \mathbb{G}$ and the internal randomness of $\mathcal{A}$.
   The CBDH assumption (with respect to $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$) states that all probabilistic polynomial time adversaries can have only negligible advantage against this CBDH problem.

2. **Decisional Bilinear Diffie-Hellman Assumption (DBDH)**
   Given a randomly chosen $P \in \mathbb{G}$, as well as aP, bP, cP for unknown, randomly chosen $a, b, c \in \mathbb{Z}_q^*$ and $Q \in \mathbb{G}_T$, it is hard to determine if $Q = e(P,P)^{abc}$ or

$Q = e(P,P)^r$ for some $r \in_R \mathbb{Z}_q^*$. Formally, an algorithm $\mathcal{A}$ has advantage ϵ in solving DBDH in $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$ if

$$Pr[\mathcal{A}(P, aP, bP, cP, e(P,P)^{abc}) = 1] - Pr[\mathcal{A}(P, aP, bP, cP, e(P,P)^r) = 1] \geq \epsilon$$

where the probability is over the random choice of $a, b, c, r \in \mathbb{Z}_q^*$, $P \in \mathbb{G}$ and the internal randomness of $\mathcal{A}$.
The DBDH assumption (with respect to $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$) states that all probabilistic polynomial time adversaries can have only negligible advantage against this DBDH problem.

3. **Strong Decisional Bilinear Diffie-Hellman Assumption (S-DBDH)**
Given a randomly chosen $P \in \mathbb{G}$, as well as aP, cP for unknown, randomly chosen $a, c \in \mathbb{Z}_q^*$ and $Q \in \mathbb{G}_T$, it is hard to determine if $Q = e(P,P)^{a^2c}$ or $Q = e(P,P)^r$ for some $r \in_R \mathbb{Z}_q^*$. Formally, an algorithm $\mathcal{A}$ has advantage ϵ in solving S-DBDH in $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$ if

$$Pr[\mathcal{A}(P, aP, cP, e(P,P)^{a^2c}) = 1] - Pr[\mathcal{A}(P, aP, cP, e(P,P)^r) = 1] \geq \epsilon$$

where the probability is over the random choice of $a, c, r \in \mathbb{Z}_q^*$, $P \in \mathbb{G}$ and the internal randomness of $\mathcal{A}$.
The S-DBDH assumption (with respect to $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$) states that all probabilistic polynomial time adversaries can have only negligible advantage against this S-DBDH problem.

2.4 Cryptographic Hash Function

A hash function or a message digest function is used to map an arbitrary length string to a fixed length (say n-bit) string. The output of a hash function is generally referred as hash value or digest of the input. A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is called a cryptographic hash function if it is efficiently computable and satisfies the following conditions:

1. **Pre-image resistance :** For a given hash value $y \in \{0,1\}^n$, it is computationally infeasible to find an input $x \in \{0,1\}^*$ such that $H(x) = y$.

2. **Second pre-image resistance :** For a given input $x \in \{0,1\}^*$, it is computationally infeasible to find an input $x' \neq x$ such that $H(x) = H(x')$.

3. **Collision resistance :** It is computationally infeasible to find inputs x and x' such that $x \neq x'$ and $H(x) = H(x')$.

Note that requirement of these conditions depends upon the role of a hash function in the underlying protocol. Until unless stated otherwise, in this thesis, a hash function is considered to be a cryptographic hash function.

2.5 Provable Security

Provable security is a part of modern cryptography which argues the security of cryptographic protocols in a formal way. In provable security, we prove that a polynomial time reduction exists between the difficulty of breaking the cryptographic protocol under a specific security model and the difficulty of solving a hard problem. The security of the protocol, possible underlying cryptographic primitives and the hard problem is parameterized in terms of the security parameter whereas adversaries are modeled using Probabilistic Polynomial time (PPT) Turing machines. Here PPT machines mean that the adversaries can be considered as abstract computational devices that use randomness and their running time is polynomially bounded by some polynomial in security parameter. The provable security paradigm[2] can be described as follows:

- **Define a cryptographic protocol formally** by specifying the behavior of each component algorithm in the protocol. Explain the inputs and outputs for these algorithms. Mention their types i.e., deterministic or probabilistic. If required, specify the correctness requirements.

- **Specify a security model** which can define the restrictions over a computationally bounded adversary and can explain what it means to break the protocol. A very common approach of specifying security models is to use a *game-based definition*. Here the security model can be considered as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. $\mathcal{A}$ can make different queries to $\mathcal{C}$ which $\mathcal{C}$ responds accordingly. The security of a cryptographic protocol with respect to the specified security model is measured in terms of the advantage of an adversary in achieving the goal specified by the security game.

- **Provide a protocol construction** which satisfies the formal definition of the cryptographic protocol.

- **Show a reduction** from the construction to an underlying primitive or some other computational hardness assumption. The reduction shows that an adversary can break the protocol with respect to the specified security model, only if it can break the underlying primitive or hardness assumption.

2.5.1 Random Oracle Model

As mentioned earlier, security models are generally stated as a game between a challenger and an adversary. Let $\mathcal{A}$ be an adversary against the cryptographic protocol that can break the protocol ("break" as defined in the security model). Now, using $\mathcal{A}$, we try to simulate another adversary $\mathcal{B}$ that can break the underlying hard problem. Note that $\mathcal{B}$ must be able to respond queries of $\mathcal{A}$ correctly. Such a reduction with no additional assumption is called a proof in *standard model*. However, designing cryptographic protocols which can be proved secure in standard model is not easy. Basically, generating a correct response for the adversary's queries is what makes the proof sometimes difficult in the standard model. To overcome this difficulty Bellare and Rogaway introduced the *Random Oracle Model* (ROM)[3]. This model assumes an existence of a random oracle H, which can be considered as a black box that responds to queries with values chosen uniformly at random from its output domain, except that for any specific query the oracle always responds with the same value every time it receives the query. Also, for an input x, the only way to get $H(x)$ is by specifically querying x to the oracle.

In practice, we instantiate the random oracles (which are assumed in the security proofs) by cryptographic hash functions.

2.6 Identity-based Non-interactive Key Exchange

The standard definition of Identity-based Non-interactive Key Exchange (ID-NIKE) was proposed by Paterson and Srinivasan in 2009 [23]. According to this definition, an ID-NIKE protocol can be described using the following three algorithms:

- **Setup**: This algorithm is executed by the Private Key Generator (PKG).

  With input security parameter 1^λ, this algorithm outputs a master public key *mpk* and a master secret key *msk*. Note that *msk* is being kept secret by the PKG to itself whilst *mpk* is publicly available to all nodes in the network.

- **Extract** ($\mathcal{E}$): This algorithm is executed by the PKG to generate secret key for the nodes in the network. With input *mpk*, *msk* and an identity $ID \in \mathcal{ID}$, this algorithm outputs a secret key SK_{ID} from some private key space $\mathcal{SK}$. Here $\mathcal{ID}$ represents the identity space.

- **Shared Key** ($\mathcal{SHK}$): This algorithm is executed by a node (say, ID_a) in the network to compute a shared secret key with another node (say, ID_b) in the network. With input mpk, the secret key of ID_a i.e., SK_{ID_a} and the identity ID_b, this algorithm returns the shared secret key $SHK(ID_a, ID_b)$ from the shared secret key space.

Correctness requirement: Here, $\mathcal{SHK}$ algorithm is required to output the same shared secret key for a pair when executed from any one of the nodes in the pair i.e., for any two identities ID_a, ID_b with corresponding secret keys, $\mathcal{SHK}$ algorithm should satisfy the following condition:

$$\mathcal{SHK}(mpk, SK_{ID_b}, ID_a) = \mathcal{SHK}(mpk, SK_{ID_a}, ID_b).$$

ID-NIKE Security Model

The standard definition of ID-NIKE security was introduced by Paterson and Srinivasan [23] in 2009. They named their security model as Indistinguishability of Shared Key (IND-SK) model. The IND-SK model is defined as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. With input security parameter 1^λ, $\mathcal{C}$ runs the **Setup** algorithm of the ID-NIKE protocol. $\mathcal{C}$ gives the master public key mpk to $\mathcal{A}$ and keeps the master secret key msk to itself. $\mathcal{A}$ then makes queries of the following three types to $\mathcal{C}$:

- **Extract:** $\mathcal{A}$ queries an identity ID_j. $\mathcal{C}$ then runs the $\mathcal{E}$ algorithm to derive the secret of ID_j i.e., SK_{ID_j} and provides it to $\mathcal{A}$.

- **Reveal:** $\mathcal{A}$ queries a pair of identities (ID_a, ID_b). $\mathcal{C}$ computes the SK_{ID_a} using the $\mathcal{E}$ algorithm as above. It then computes the shared secret key $\mathcal{SHK}(mpk, SK_{ID_a}, ID_b)$ and provides it to $\mathcal{A}$.

- **Test:** $\mathcal{A}$ provides two challenge identities ID_x and ID_y. $\mathcal{C}$ responds this by tossing an unbiased coin $\delta \in_R \{0, 1\}$. If $\delta = 1$ then $\mathcal{C}$ gives $\mathcal{SHK}(mpk, SK_{ID_x}, ID_y)$ to $\mathcal{A}$; otherwise, $\mathcal{C}$ gives a random element from the shared key space to $\mathcal{A}$.

$\mathcal{A}$ can make arbitrary but polynomial number of **Extract** and **Reveal** queries. However, $\mathcal{A}$ is allowed to make only one test query. Also the queries can be made adaptively i.e., $\mathcal{A}$ can make **Extract**, **Reveal** and **Test** query in any order. However, the following restrictions are required on the adversary to to prevent it from winning the security game trivially:

1. $\mathcal{A}$ is not allowed to make **Extract** queries on the test identities.

2. $\mathcal{A}$ is not allowed to make **Reveal** query on the test identities (in either order).

In the end, $\mathcal{A}$ outputs a bit δ', and wins the security game if $\delta = \delta'$. The advantage of the adversary $\mathcal{A}$ in the IND-SK model is given as:

$$Adv_{\mathcal{A}}^{\text{IND-SK}}(\lambda) = |Pr[\delta = \delta'] - 1/2|.$$

We say that an ID-NIKE protocol is IND-SK secure if for any polynomial time adversary $\mathcal{A}$, the function $Adv_{\mathcal{A}}^{\text{IND-SK}}(\lambda)$ is negligible.

2.6.1 Sakai-Ohgishi-Kasahara Key Exchange Protocol (SOK)

The first identity based key exchange protocol (named as SOK) was proposed by Sakai, Ohgishi and Kasahara in 2000 [25]. SOK protocol uses bilinear pairing for the key exchange. It is non-interactive and fully resilient against arbitrary number of node compromising. However it does not support hierarchical or decentralized networks. We define SOK protocol using the following three algorithms:

- **Setup:** This algorithm is executed by the Private Key Generator (PKG).

  With input security parameter 1^λ, this algorithm generates two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order q, a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ and a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$. Now, this algorithm chooses a random generator P of the group $\mathbb{G}$ and a random element $s \in_R \mathbb{Z}_q^*$. It then outputs the master public key mpk as $(\mathbb{G}, \mathbb{G}_T, e, q, P, P_0 = sP, H)$ and the master secret key msk as s. Note that s (msk) is kept secret by the PKG to itself whilst mpk is published to all nodes in the network. The identity space is defined here as $\mathcal{ID} = \{0,1\}^*$ (bit-strings of arbitrary lengths) whereas secret key space and shared-secret key space are $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

- **Extract:** This algorithm is executed by the PKG to generate secret key for the nodes in the network. With input mpk, msk and an identity $ID \in \mathcal{ID}$, this algorithm outputs the secret key $SK_{ID} = s \cdot H(ID)$.

- **Shared Key:** This algorithm is executed by a node (say, ID_a) to compute a shared secret key with another node (say, ID_b). With input mpk, the secret key of ID_a i.e., SK_{ID_a} and the identity ID_b, this algorithm returns the shared secret key $SHK(ID_a, ID_b) = e\big(SK_{ID_a}, H(ID_b)\big)$.

Correctness: Correctness of the **Shared Key** algorithm can be verified using the bilinear property of pairing e i.e.,

$$\begin{aligned} SHK_{A,B} = e\big(S_{ID_A}, H(ID_B)\big) &= e\big(s \cdot H(ID_A), H(ID_B)\big) \\ &= e\big(H(ID_A), s \cdot H(ID_B)\big) = e\big(H(ID_A), S_{ID_B}\big) = SHK_{B,A}. \end{aligned}$$

Security of SOK-ID-NIKE Protocol

Theorem 2.1 *SOK-ID-NIKE protocol is secure assuming DBDH assumption to be hard in groups $(\mathbb{G}, \mathbb{G}_T, e)$. In other words, for any IND-SK adversary $\mathcal{A}$ against SOK-ID-NIKE protocol that makes q queries to hash function H, there is an algorithm $\mathcal{B}$ that solves the DBDH problem in $(\mathbb{G}, \mathbb{G}_T, e)$ with*

$$Adv_{\mathcal{B}}^{DBDH} = Adv_{\mathcal{A}}^{\text{SOK-ID-NIKE}}(\lambda)/q_H^2.$$

The claim is in the random oracle model, i.e. H should be modeled here as a random oracle.

Proof: Let us consider $\mathcal{A}$ as an adversary with advantage ϵ against SOK-ID-NIKE protocol. We show how to construct an algorithm (a.k.a. simulator) $\mathcal{B}$ out of $\mathcal{A}$ which can solve the DBDH problem in $(\mathbb{G}, \mathbb{G}_T, e)$.

Let (aP, bP, cP, Q) be the instance of a DBDH problem in $(\mathbb{G}, \mathbb{G}_T, e)$ and $\mathcal{B}$ is given the input as $(\mathbb{G}, \mathbb{G}_T, e, q, P, aP, bP, cP, Q)$ with a task to distinguish whether $Q = e(P,P)^{abc}$ or $Q \in_R \mathbb{G}_T$.

$\mathcal{B}$ first simulates $\mathcal{A}$ with the IND-SK experiment. It gives $\mathcal{A}$ the mpk as $(\mathbb{G}, \mathbb{G}_T, e, P, P_0 = cP, H)$. Here H is a random oracle controlled by $\mathcal{B}$. Let q_H be the total number of hash queries done by $\mathcal{A}$ during the simulation. $\mathcal{B}$ chooses two indices I and J uniformly at random from $\{1, 2, \ldots, q_H\}$. Now, $\mathcal{A}$ can make relevant queries which $\mathcal{B}$ responds as follows:

- **Hash queries:** $\mathcal{A}$ queries an identity ID from the identity space. To Respond these queries $\mathcal{B}$ maintains an H-table with entries of the form $(ID, r, H(ID))$. If the queried ID is already there in the H-table, $\mathcal{B}$ responds with corresponding $H(ID)$ value. Otherwise, $\mathcal{B}$ responds to the i-th query of $\mathcal{A}$ as follows:

    1. if $i = I$, then $\mathcal{B}$ adds entry $(ID_I, \perp, aP)$ to the H-table and returns $H(ID_I) = aP$.

2. if $i = J$, then $\mathcal{B}$ adds entry $(ID_J, \perp, bP)$ to the H-table and returns $H(ID_J) = bP$.
3. Otherwise, $\mathcal{B}$ chooses r_i uniformly at random from $\mathbb{Z}_q^*$, adds (ID_i, r_i, r_iP) to the H-table, and returns $H(ID_i) = r_iP$.

- **Extract queries:** $\mathcal{A}$ queries an identity ID. $\mathcal{B}$ checks the H-table for the entry corresponding to ID. If this hash query wasn't done before then $\mathcal{B}$ makes an **Hash query** on ID. If $ID \in \{ID_I, ID_J\}$ then $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ finds the entry $(ID, r, H(ID))$ in the H-table and responds with $SK_{ID} = r(cP)$.

- **Reveal queries:** $\mathcal{A}$ queries an identity pair $\{ID_i, ID_j\}$. $\mathcal{B}$ checks the H-table for the entry corresponding to ID_i and ID_j. If these hash queries wasn't done before then $\mathcal{B}$ makes the **Hash queries** on ID_i and ID_j. $\mathcal{B}$ now responds to the query of $\mathcal{A}$ in the following manner:

  1. If $\{ID_i, ID_j\} = \{ID_I, ID_J\}$, then $\mathcal{B}$ aborts the simulation.
  2. If $|\{ID_I, ID_J\} \cap \{ID_i, ID_j\}| = 1$ then $\mathcal{B}$ first finds the $r \in \{r_i, r_j\}$ such that $r \neq \perp$. It then responds with -

  $$SHK(ID_i, ID_j) = \begin{cases} e(H(ID_i), r(cP)), & \text{if } r = r_j \\ e(H(ID_j), r(cP)), & \text{if } r = r_i \end{cases}$$

  3. If $|\{ID_I, ID_J\} \cap \{ID_i, ID_j\}| = 0$, then $\mathcal{B}$ responds the query with -

  $$SHK(ID_i, ID_j) = e(H(ID_i), r_j(cP)).$$

- **Test Query:** $\mathcal{A}$ provides two challenge identities ID_x and ID_y. If $\{ID_x, ID_y\} \neq \{ID_I, ID_J\}$ then $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ returns Q as the shared secret key for the challenge identities.

In the end, $\mathcal{A}$ responds with a guess bit $\delta \in \{0, 1\}$. Note that if the DBDH instance really contained $e(P, P)^{abc}$ as Q, then -

$$\begin{aligned} Q = e(P, P)^{abc} &= e(aP, c(bP)) \\ &= e(H(ID_I), msk \cdot H(ID_J)) \\ &= e(H(ID_I), SK_{ID_J}) = SHK(ID_I, ID_J). \end{aligned}$$

i.e., Q corresponds to the real shared secret key between the challenge identities. Therefore, in this case, $\delta = 1$ in the simulation of $\mathcal{A}$ under $\mathcal{B}$. On the other hand, if $Q = e(P,P)^r$, for some random $r \in \mathbb{Z}_q^*$, then the test query response by $\mathcal{B}$ is a random element from $\mathbb{G}_T$ which corresponds to $\delta = 0$.

In other words, δ is a correct response for the DBDH problem if it is a correct response in the corresponding IND-SK experiment. Which means for the events when $\mathcal{B}$ does not abort the simulation, the advantage of $\mathcal{B}$ against DBDH problem is same as the advantage of $\mathcal{A}$ against the SOK-ID-NIKE.

Analysis: Let us consider $\mathcal{U}$ to be the event when $\mathcal{B}$ does not abort the simulation. It is easy to see that $Pr(\mathcal{U}) \geq 1/q_H^2$. Thus if the advantage of $\mathcal{A}$ against SOK-ID-NIKE protocol is ϵ, then the advantage of $\mathcal{B}$ against DBDH problem can be given as:

$$\epsilon'(\lambda) \geq \frac{\epsilon(\lambda)}{q_H^2}.$$

Here q_H is a polynomial in λ. Which means, if $\epsilon(\lambda)$ is a non-negligible function then we can say that $\epsilon'(\lambda)$ is also non-negligible. $\square$

2.6.2 Basic ID-based One-way-function Scheme (BIOS)

Basic Id One-way function Scheme (BIOS) was proposed by Lee and Stinson in 2003 [20]. BIOS is a deterministic key pre-distribution scheme. It is also non-interactive, identity-based and fully resilient against arbitrary number of node compromising. The original description of this protocol is based on Euler graphs. A simpler approach for BIOS was later reported in [27]. This approach is based on modular arithmetic. It is easy to understand and is better for implementing purposes. We define BIOS (with modular arithmetic approach) using the following three algorithms:

- **Setup:** This algorithm is executed by the PKG.

  With input security parameter 1^λ and the number of nodes in the network (n), this algorithm chooses a prime q and generates the msk as a key-ring $\mathcal{K} = \{K_0, K_1, K_2, \ldots\ldots, K_{n-1}\}$. Here n is assumed to be odd and each key $K_i \in \mathbb{Z}_q^*$ corresponds to the i^{th} identity (ID_i) in the network. The algorithm also outputs mpk as (q, h) where $h : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ is a hash function.

- **Key Distribution ($\mathcal{KD}$):** This algorithm is executed by the PKG.

With *msk* and a user's identity ID_j as input, this algorithm outputs a set containing $\lceil \frac{n}{2} \rceil$ elements from $\mathbb{Z}_q^*$ (as shown below in the Table 2.1). We define this set as the secret key set SK_{ID_j}.

Nodes (ID_j)	Secret-Key sets (SK_{ID_j})					
ID_0	K_0	$h(K_{\lceil \frac{n}{2} \rceil} \Vert ID_0)$	$h(K_{\lceil \frac{n}{2} \rceil+1} \Vert ID_0)$	$\cdots$	$h(K_{n-2} \Vert ID_0)$	$h(K_{n-1} \Vert ID_0)$
ID_1	K_1	$h(K_{\lceil \frac{n}{2} \rceil+1} \Vert ID_1)$	$h(K_{\lceil \frac{n}{2} \rceil+2} \Vert ID_1)$	$\cdots$	$h(K_{n-1} \Vert ID_1)$	$h(K_0 \Vert ID_1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ID_{n-1}	K_{n-1}	$h(K_{\lceil \frac{n}{2} \rceil-1} \Vert ID_{n-1})$	$h(K_{\lceil \frac{n}{2} \rceil} \Vert ID_{n-1})$	$\cdots$	$h(K_{n-3} \Vert ID_{n-1})$	$h(K_{n-2} \Vert ID_{n-1})$

Table 2.1 Key distribution in BIOS

To understand the pattern of this table, let us define a term called "window size" as $\omega = \lfloor \frac{n}{2} \rfloor$. The key distribution is done in such a way that the node ID_j can compute the shared secret keys for node $ID_{(j+1) \mod n}$ to node $ID_{(j+\omega) \mod n}$ (i.e. exactly ω nodes). However, for remaining $\big[(n-1)-\omega\big]$ nodes, ID_j will have to store the shared secret keys. The total number of keys stored per node (including the key K_j) will be -

$$|SK_{ID_j}| = (n-1) - \omega + 1 = (n - \omega) = \left(n - \left\lfloor \frac{n}{2} \right\rfloor\right) = \left\lceil \frac{n}{2} \right\rceil.$$

- **Shared Key ($\mathcal{SHK}$):** This algorithm is executed by a node (say, ID_a) to compute a shared secret key with another node (say, ID_b). With input the secret key set SK_{ID_a} and the identity ID_b, this algorithm outputs the shared secret key between ID_a and ID_b in the following manner:

$$\mathcal{SHK}(ID_a, ID_b) = \begin{cases} h(K_a || ID_b) \text{ if } (b-a) \mod n \le \omega, \\ h(K_b || ID_a) \text{ if } (b-a) \mod n > \omega. \end{cases}$$

Correctness: The correctness of this shared key algorithm can be verified by the fact that if $(b-a) \mod n \le \omega$ then the true shared key $h(K_a||ID_b) \in SK_{ID_b}$ which is computed by ID_a is already stored in ID_b's storage. Similarly, if $(b-a) \mod n \le \omega$ then the true shared key $h(K_b||ID_a) \in SK_{ID_a}$ which is computed by ID_b is already stored in ID_a's storage.

Remark - We can easily notice that the BIOS protocol works correctly only when the number of nodes in the network is odd. However, new node admissions can make the size of the network even. In such conditions, we add a dummy node. The shared secret keys of any node in the network with the dummy node is of no use and when a new node joins this network again, we replace the dummy node by this new node.

Security of BIOS

A brief security analysis of BIOS was given by Lee and Stinson in [20]. A formal security proof for BIOS was later provided by Tiwari in [27] using an assumption called "Direct Computation of Secret Key". This assumption is considered as a necessary requirement for the security of any identity-based protocol.

Direct Computation of Secret Key (DCSK)

The security of any identity-based protocol depends upon an inherent underlying assumption which says that the direct computation of a secret key for an uncompromised identity is hard. If an adversary can directly compute the secret key of an uncompromised node then it can always break the corresponding identity-based protocol. Therefore, we can say that DCSK is a necessary (but may not be sufficient) condition for the security of any identity-based protocol.

DCSK can be represented as a security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The challenger $\mathcal{C}$ first give the public parameters to the adversary $\mathcal{A}$. The adversary is allowed to make arbitrary (but polynomial) number of Extract queries. In these queries, $\mathcal{A}$ provides an identity to $\mathcal{C}$, which $\mathcal{C}$ responds with the corresponding secret key for the queried identity. In the end, $\mathcal{A}$ returns a secret key for an uncompromised identity.

DCSK assumption states that all polynomial time adversaries have only negligible advantage against DCSK i.e.,

$$Adv_{\mathcal{A}}^{DCSK} \leq \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is a negligible function.

BIOS is an identity-based protocol, therfore, DCSK assumption is required. However, we show that DCSK is a sufficient assumption for BIOS to be secure in the COMP-SK model. COMP-SK is a weaker version of the IND-SK model. In COMP-SK model, there is no **Test** query and to win the game, $\mathcal{A}$ is required to output the actual shared secret key $SHK(ID_a, ID_b)$ between two identities ID_a, ID_b which are neither queried in **Extract** query nor together queried in **Reveal** query.

Theorem 2.2 *BIOS is secure under COMP-SK model, where h is modeled as random oracle, provided DCSK is hard to break.*

Proof:
Let us consider $\mathcal{A}$ be the adversary for BIOS. We show how to construct another adversary $\mathcal{B}$ out of $\mathcal{A}$ which is capable to break DCSK. Let $\mathcal{C}$ be the challenger for BIOS. $\mathcal{C}$ runs the "Setup" algorithm of BIOS and generates the mpk and the key-ring $\mathcal{K} = \{K_0, \ldots, K_{n-1}\} \subset \mathbb{Z}_q^*$. The mpk is provided to $\mathcal{A}$, however, the hash function h is controlled by $\mathcal{C}$ as a random oracle. $\mathcal{A}$ can make relevant queries which $\mathcal{C}$ responds as follows:

- **Hash queries:** $\mathcal{A}$ provides a query of the form $(K_i||ID_j)$. To respond this query $\mathcal{C}$ maintains an h-table with entries as $(K_i||ID_j, h(K_i||ID_j))$. If the queried element is already there in the h-table, $\mathcal{C}$ responds with the corresponding hash values. Otherwise, $\mathcal{C}$ chooses a random element r from $\mathbb{Z}_q^*$, adds this entry in the h-table and gives r to $\mathcal{A}$.

- **Extract queries:** $\mathcal{A}$ queries an identity ID to $\mathcal{C}$. $\mathcal{C}$ checks the h-table for all required entries and if needed makes new entries for ID. $\mathcal{C}$ now uses its key-ring and h to generate a set of $\lceil \frac{n}{2} \rceil$ random elements from $\mathbb{Z}_q^*$. $\mathcal{C}$ responds this set to $\mathcal{A}$.

- **Reveal queries:** $\mathcal{A}$ queries an identity pair (ID_i, ID_j) to $\mathcal{C}$. $\mathcal{C}$ checks the h-table for all required entries and if needed makes new entries which are relevant to ID_i and ID_j. $\mathcal{C}$ now uses its key-ring and h to choose a random element from $\mathbb{Z}_q^*$. $\mathcal{C}$ responds this element as the shared secret key to $\mathcal{A}$.

$\mathcal{A}$ now outputs a shared secret key $h(K_i^*||ID_j^*)$ between two identities ID_i^* and ID_j^* which are neither queried in the **Extract** query, nor together queried in the **Reveal** query.

Note that h is modeled here as a random oracle. Thus $\mathcal{A}$ can output a correct shared secret key between ID_i^* and ID_j^* (i.e., $h(K_i^*||ID_j^*)$) if and only if $\mathcal{A}$ has some way to compute the key K_i^*. In this simulation, these are the only two remaining ways which $\mathcal{A}$ can use to compute the secret key K_i^*:

1. $\mathcal{A}$ can use some **Extract** or **Reveal** query outputs which contains K_i^* as a part of their input (there are $\sim \frac{n}{2}$ possible shared keys in the network which contains K_i^* as a part of their input). However, these queries' outputs are of the form $h(K_i^*||ID)$ and since h is simulated as a true random function, finding K_i^* back from it is hard.

2. $\mathcal{A}$ can break DCSK, i.e., it can directly compute K_i^* without compromising ID_i^* and then by querying $(K_i^*||ID_j^*)$ in **Hash** query it can output the correct shared key. Which means this adversary with non-negligible advantage against BIOS can act as an adversary against DCSK with same advantage.

□

Chapter 3

Hierarchical Identity-based Non-interactive Key Exchange

3.1 H-ID-NIKE Definition

The definition for Hierarchical Identity-based Non-interactive Key Exchange was initially proposed by Freire in 2014 [12]. However, the corresponding security model for this definition contains an additional "non-trivial" restriction that the adversary can not choose an ancestor-descendant pair as the target pair. Later, in 2017, Tiwari [27] provided a modified version of this definition. In this thesis, we have used this modified definition as the H-ID-NIKE definition.

Let us consider $\mathcal{ID}$ as the identity space and $\mathcal{SK}$ as the secret key space. Any node in the system can be represented by $ID^i_j \in \mathcal{ID}^i \subseteq \mathcal{ID}$ where i is the level of the node and j is the rank of the node in the hierarchy. Any H-ID-NIKE protocol for an ℓ-levelled hierarchy (with a root node or PKG at 0^{th} level) can be described as follow:

- **Setup :** This algorithm is executed by the PKG.

  With input security parameter 1^λ and hierarchy depth $\ell \in \mathbb{N}$, this algorithm computes a master secret key msk and a master public key mpk. Note that msk is being kept secret by the PKG to itself whilst mpk is publicly available to all nodes in the network.

- **Key Delegation ($\mathcal{D}$) :** This algorithm is executed by a node (say, ID^L_m) at level $L \in [0, \ell - 1]$ in the hierarchy to generate secret keys for one of its child nodes (say, ID^{L+1}_j). With input master public key mpk, secret key set of the parent node $SK_{ID^L_m}$ and the identity of child node ID^{L+1}_j, this algorithm outputs the secret key set for node ID_j^{L+1} i.e.,

$$\mathcal{D}(mpk, SK_{ID_m^L}, ID_j^{L+1}) = SK_{ID_j^{L+1}}.$$

We can easily see that any ancestor of ID_j^{L+1} at level t can also compute the secret key set of ID_j^{L+1} by continually applying the $\mathcal{D}$ algorithm from level t to level L. Note that for the first level nodes key delegation, msk is being used as SK_{root}.

- **Shared Key Computation ($\mathcal{SHK}$) :** This algorithm is executed by a node (say, $ID_b^{L_1}$) at level $L_1 \in [1, \ell]$ to generate a shared secret key with another node (say, $ID_a^{L_2}$) at level $L_2 \in [1, \ell]$ in the hierarchy. With input master public key mpk, secret key set $SK_{ID_b^{L_1}}$ and the identity of the other node $ID_a^{L_2}$, this algorithm computes a shared secret key as $\mathcal{SHK}(mpk, SK_{ID_b^{L_1}}, ID_a^{L_2})$.

Correctness requirement : Here, $\mathcal{SHK}$ algorithm is required to output the same shared secret key for a pair when executed from any one of the nodes in the pair i.e., for any two identities $ID_b^{L_1}, ID_a^{L_2}$ with corresponding secret keys, $\mathcal{SHK}$ algorithm should satisfy the following condition:

$$\mathcal{SHK}(mpk, SK_{ID_b^{L_1}}, ID_a^{L_2}) = \mathcal{SHK}(mpk, SK_{ID_a^{L_2}}, ID_b^{L_1}).$$

3.2 H-ID-NIKE Security Model

The standard definition of ID-NIKE security was introduced by Paterson and Srinivasan [23] in 2009. They named their security model as Indistinguishability of Shared Key (IND-SK) model. Later, in 2014, Freire proposed an hierarchical version of this IND-SK model [12]. However, there is an additional "non-trivial" restriction in their model due to the underlying H-ID-NIKE definition. We refer Freire's model as a weaker variant of H-IND-SK model. Another variant of H-IND-SK model was later presented by Tiwari in [27] which is called as M-H-IND-SK model. M-H-IND-SK model also contains some restrictions which are not essential. We hereby describe the general, adaptive H-IND-SK model and explain how it strengthens the models in [12, 27]

The H-IND-SK model is defined as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. With input security parameter 1^λ and hierarchy depth $\ell \in \mathbb{N}$, $\mathcal{C}$ runs the **Setup** algorithm of the H-ID-NIKE protocol. $\mathcal{C}$ gives the master public key mpk to $\mathcal{A}$ and keeps the master secret key msk to itself. $\mathcal{A}$ then makes queries of the following three types to $\mathcal{C}$:

- **Extract:** $\mathcal{A}$ queries an identity ID_j^L. $\mathcal{C}$ then runs the $\mathcal{D}$ algorithm to derive the secret of ID_j^L i.e., $SK_{ID_j^L}$ and provides it to $\mathcal{A}$.

- **Reveal:** $\mathcal{A}$ queries a pair of identities $(ID_b^{L_1}, ID_a^{L_2})$ to $\mathcal{C}$. $\mathcal{C}$ first computes the $SK_{ID_b^{L_1}}$ using the $\mathcal{D}$ algorithm as above. It then computes the shared secret key $\mathcal{SHK}(mpk, SK_{ID_b^{L_1}}, ID_a^{L_2})$ and provides it to $\mathcal{A}$.

- **Test:** $\mathcal{A}$ provides two challenge identities $ID_x^{L_3}$ and $ID_y^{L_4}$. $\mathcal{C}$ responds this by tossing an unbiased coin $\delta \in_R \{0, 1\}$. If $\delta = 1$ then $\mathcal{C}$ gives $\mathcal{SHK}(mpk, SK_{ID_x^{L_3}}, ID_y^{L_4})$ to $\mathcal{A}$; otherwise, $\mathcal{C}$ gives a random element from the shared key space to $\mathcal{A}$.

$\mathcal{A}$ can make arbitrary but polynomial number of **Extract** and **Reveal** queries. However, $\mathcal{A}$ is allowed to make only one test query. Also the queries can be made adaptively i.e., $\mathcal{A}$ can make **Extract**, **Reveal** and **Test** query in any order. However, the following restrictions are required on the adversary to to prevent it from winning the security game trivially:

1. $\mathcal{A}$ is not allowed to make **Extract** queries on the test identities or any of their ancestors.

2. $\mathcal{A}$ is not allowed to make **Reveal** query on the test identities (in either order).

In the end, $\mathcal{A}$ outputs a bit δ', and wins the security game if $\delta = \delta'$. The advantage of the adversary $\mathcal{A}$ in the H-IND-SK model is given as:

$$Adv_{\mathcal{A}}^{\text{H-IND-SK}}(\lambda) = |Pr[\delta = \delta'] - 1/2|.$$

We say that an H-ID-NIKE protocol is H-IND-SK secure if for any polynomial time adversary $\mathcal{A}$, the function $Adv_{\mathcal{A}}^{\text{H-IND-SK}}(\lambda)$ is negligible.

3.3 Comparison with Existing Models

Our H-IND-SK model strengthens all three existing security models presented in [12, 16, 27]. Here are the differences between our model and the mentioned models :

1. In our H-IND-SK model, test nodes can be chosen in a manner such that one is ancestor of the other. However, in Freire's model, adversary is restricted from doing this. This restriction occurs in Freire's model due to their underlying definition of H-ID-NIKE where two nodes do not compute a shared secret key if either of them is an ancestor of the other.

2. The M-H-IND-SK model does not allow **Reveal** queries on the ancestors of the target nodes. However, in our H-IND-SK model we allow all such queries.

3. The M-H-IND-SK model does not allow the adversary to make **Extract** queries on the left most descendants of the higher levelled target node up to the parent level of the lower level target node. This restriction occurs in M-H-IND-SK model due to their underlying definition of H-ID-NIKE where these left most descendants of the higher levelled target node can compute the shared secret key of the target pair. However, there is no such restriction in our H-IND-SK model.

4. Unlikely to our H-IND-SK, the model given by Gennaro *et al.* does not allow the adversary to make any kind of **Reveal** queries. Therefore this model does not capture one of the basic security requirements which is an adversary, even after compromising the shared secret keys for some pairs of nodes, should not be able to compute the shared secret key between any two uncompromised nodes. The importance of the adversarial access to the **Reveal** queries is closely discussed by Paterson and Srinivasan in [23].

5. The model given by Gennaro *et al.* forces an adversary to make all of its non-leaf node **Extract** queries before compromising a leaf level node. This restricts the adaptiveness of an adversary and the possibilities of attacks. On the other hand, in our model, there is no such restriction on the adversary.

6. In the model of Gennaro *et al.*, target nodes can only be chosen from the leaf level. Whereas in our H-IND-SK model, the target nodes can be at any level of the hierarchy.

3.4 Hybrid Hierarchical Key Agreement Scheme

Hybrid Hierarchical Key Agreement Scheme (HHKAS) was introduced in 2008 by Gennaro *et al.* [16] as a key exchange protocol for MANETs. This protocol is a hybrid of two protocols, subset based key agreement scheme (SKAS)[11] and the SOK[25] key exchange protocol. HHKAS is a hierarchical, identity-based and non-interactive protocol. It is also fully resilient against any number of node compromising at the leaf level. However, it is only resilient up to a threshold number of node compromising in the non-leaf levels. Another important point to note in HHKAS is that the key exchange is guaranteed and can be shown secure at the leaf level only.

3.4.1 Construction of HHKAS

We can define HHKAS using the following algorithms:

- **Setup**: This algorithm is executed by the PKG (root node). With the number of keys in the root's key-ring M and the probabilities $p_i \in (0,1)$ for each level i in the hierarchy as input the algorithm outputs msk as a key-ring with M randomly chosen secret keys for the root. These keys are chosen from $\mathbb{Z}_q^*$ for a large prime q. Here p_i determines the fraction of an i^{th} level node's key-ring which is going to be forwarded to its children. This algorithm also runs the Setup algorithm of SOK protocol and publishes the mpk which includes two cyclic groups $\mathbb{G}, \mathbb{G}_T$ of prime order q, a hash function $H : \{0,1\}^* \to \mathbb{G}$ and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Finally, this algorithm outputs another public hash function $h : \{0,1\}^* \times \{x \mid 1 \leq x \leq M\} \to (0,1)$ where x is an array which denotes the publicly known indices of root's key-ring.

- **Key Delegation at level L ($\mathcal{D}_L$; $L \neq \ell$)**: This algorithm is executed by a node at level $L-1$ to define secret keys for its children at level L. Let ID_m^{L-1} is a node at level $L-1 \in [0, \ell-2]$ in the hierarchy who wants to delegate secret keys to its child ID_j^L. With input the key-ring of ID_m^{L-1} as $R_m^{L-1} = \{K_1, K_2, \ldots\}$ and the hash function h, the algorithm outputs the key-ring of ID_j^L as -

$$R_j^L = \{K_x \in R_m^{L-1} \mid h(ID_m^{L-1}, x) < p_L\} \text{ where } 1 \leq x \leq M.$$

- **Key Delegation at level ℓ ($\mathcal{D}_\ell$)**: This algorithm is executed by a node at level $\ell - 1$ to define secret keys for its children in the leaf level. Let $ID_m^{\ell-1}$ is a node at level $\ell - 1$ in the hierarchy who wants to delegate secret keys to its child ID_j^ℓ. With input the key-ring of $ID_m^{\ell-1}$ as $R_m^{\ell-1} = \{K_1, K_2, \ldots, K_n\}$ and the hash function H, the algorithm outputs the secret key set of ID_j^ℓ as -

$$\mathcal{K}_j^\ell = \{K_i \cdot H(ID_m^{\ell-1}) \in \mathbb{G} \mid 1 \leq i \leq n\}.$$

- **Shared Key Computation at level ℓ ($\mathcal{SHK}$)**: This algorithm is executed by a leaf level node to compute a shared secret key with any other leaf level node. Let a leaf node ID_a^ℓ wants to compute a shared secret key with another leaf node ID_b^ℓ. With input the secret key set $\mathcal{K}_a^\ell$ and the identity ID_b^ℓ, this algorithm repeats the hash calculations from the root level to the second last level to determine the intersection of ID_a^ℓ and ID_b^ℓ's parents' key rings. Note that the secret key set of leaf nodes contains elements from $\mathbb{G}$ corresponding to the elements of their parents' key rings. Thus the algorithm can determine the intersection between ID_a^ℓ and ID_b^ℓ's secret key sets. Let the set of root's key-ring indices for which both the leaf nodes got the secret keys be I then the algorithm outputs the shared secret key between ID_a^ℓ and ID_b^ℓ as -

$$\mathcal{SHK}(ID_a^\ell, ID_b^\ell) = e(K \cdot H(ID_a^\ell), H(ID_b^\ell)) \text{ where } K = \sum_{i \in I} K_i.$$

Correctness: The correctness of the shared secret keys can be argued using the bilinear property of the pairing e i.e.,

$$\begin{aligned}\mathcal{SHK}(ID_a^\ell, ID_b^\ell) &= e(K \cdot H(ID_a^\ell), H(ID_b^\ell)) \\ &= e(H(ID_a^\ell), K \cdot H(ID_b^\ell)) = e(K \cdot H(ID_b^\ell), H(ID_a^\ell)) \\ &= \mathcal{SHK}(ID_b^\ell, ID_a^\ell)\end{aligned}$$

The probability p_i defined for the level i depends upon the threshold value t_i associated with that level in the hierarchy. In order to withstand up to t_i number of node compromises at level i, the optimal value for the parameter p_i is $1/(t_i + 1)$. The number of keys in root's key-ring can be determined based on hierarchy depth ℓ, probability values p_i, threshold values t_i, the number of nodes in the network N and a security parameter m as -

$$\begin{aligned} M &= \left\{ \frac{m}{\prod_{i=1}^{\ell-1} {p_i}^2 (1-p_i)^{t_i}} \right\} + 2\log N \\ &\approx m e^{\ell-1} \cdot \prod_{i=1}^{\ell-1} t_i(t_i+1) + 2\log N. \end{aligned}$$

Here m ensures that an attacker against the protocol which compromises up to t_i number of nodes in each level i will not have probability more than e^{-m} of learning a shared key between two uncompromised nodes.

3.4.2 Limitations in HHKAS

There are certain limitations in the HHKAS which are described as follows -

1. The protocol description of HHKAS provides key agreement only at leaf levels. Freire [12] has described this limitation in HHKAS while discussing their definition of an H-ID-NIKE protocol.

2. In HHKAS, key agreement at leaf level is not 100% guaranteed due to the randomized key pre-distribution in the non-leaf levels. In other words, there is a possibility for existence of some nodes with no common elements in their key-rings and therefore their corresponding leaf level nodes will not be able to compute a shared secret key between them.

3. HHKAS has very bad scalability. Increasing the hierarchy depth or thresholds by some moderate values, makes the shared key computation times and the storage requirements impractical for the real life applications. To support this claim, we have provided implementation results and comparative analysis of HHKAS with our proposed solution in the Chapter 5.

4. As discussed in section 3.3, the underlying security model of HHKAS is a weaker model. It neither allows the adversary to make any Reveal query nor it allows the adversary to choose its target nodes from a non-leaf level in the hierarchy.

Remark - For applications which requires full resilience and inter-level key exchange, HHKAS is not a good solution. This is because, with full resilience (i.e., $t_i = N_i \geq b^i$ where N_i denotes the number of nodes at level i and $b \geq 2$ is the minimum branching factor in the hierarchy), the storage requirement of any node in the hierarchy grows exponentially with the depth of the hierarchy. The key storage of a leaf node K_{leaf} in a fully resilient HHKAS network can be derived as shown below -
for B as the branching factor at second last level in the hierarchy, we have -

$$\begin{aligned} N &\geq \sum_{i=1}^{\ell-1} b^i + b^{\ell-1}B = \left[\frac{b(b^{\ell-1}-1)}{b-1} + b^{\ell-1}B\right] \\ \implies N &\geq b^{\ell-1}\left[(1-b^{-1})^{-1} + B\right] - 2 \\ \implies b^{\ell-1} &\geq \left[\frac{N+2}{((1-b^{-1})^{-1}+B)}\right]. \end{aligned}$$

$$\begin{aligned} \text{Now, since } K_{root} &= me^{\ell-1} \cdot \prod_{i=1}^{\ell-1} t_i(t_i+1) + 2\log N \\ \implies K_{root} &\geq me^{\ell-1} \cdot \prod_{i=1}^{\ell-1} t_i(t_i+1) \end{aligned}$$

$$\text{therefore, } K_{leaf} = \left\lfloor \frac{K_{root}}{\prod_{i=1}^{\ell-1}(t_i+1)} \right\rfloor \geq me^{\ell-1} \cdot \prod_{i=1}^{\ell-1} t_i$$

$$\implies K_{leaf} \geq \quad me^{\ell-1} \cdot \prod_{i=1}^{\ell-1} b^i \quad \geq \quad me^{\ell-1} \cdot (b^{\ell-1})^{\ell/2}$$

$$\implies K_{leaf} \geq \quad me^{\ell-1} \cdot \left\lfloor \frac{N+2}{((1-b^{-1})^{-1}+B)} \right\rfloor^{\ell/2}$$

We can notice that in a hierarchy with N number of nodes, the key storage of a leaf node with fully resilient HHKAS is $\mathcal{O}(N^{\ell/2})$ which is worse than the trivial approach of secret key distribution (discussed in Section 3.5). This bad scalability of HHKAS was one of the main motivation for the new solution presented in this thesis.

3.5 Trivial Approach

In this section, we present a formal description of a trivial approach to distribute secret keys in a hierarchical network. This trivial distribution is non-interactive, identity-based, hierarchical and fully resilient against arbitrary number of node compromising.

3.5.1 Construction of the Trivial Solution

We can define the Trivial solution using the following algorithms:

- **Setup**: This algorithm is executed by the PKG (root node). With security parameter 1^λ and the number of nodes in the hierarchy N as input, this algorithm outputs *msk* as a key-ring with ${}^{N}C_2$ randomly chosen secret keys for the root. These keys are chosen from $\mathbb{Z}_q^*$ for a large prime q. Each one of these secret keys corresponds to a unique pair of nodes in the hierarchy. In other words, the key-ring of the root can be written as -

$$\mathcal{R}_{root} = \{K_{AB} \mid A, B \in \mathcal{ID} \text{ and } A \neq B\}$$

  Here $\mathcal{ID}$ represents the identity space and $K_{AB} = K_{BA}$.

- **Key Delegation at level L ($\mathcal{D}_L$)**: This algorithm is executed by a node at level $L-1$ to delegate secret keys for its children at level L. Let ID_m^{L-1} is a node at level $L-1$ in the hierarchy who wants to delegate secret keys to its child ID_j^L.

With input the key-ring of ID_m^{L-1} as $\mathcal{R}_m^{L-1} = \{K_{AB} \mid A \vee B \in \mathcal{ID}_{m(L-1)}\}$, the algorithm outputs the key-ring of ID_j^L as -

$$\mathcal{R}_j^L = \{K_{AB} \mid A \vee B \in \mathcal{ID}_{jL}\}.$$

Here $\mathcal{ID}_{xy}$ represents the identity space of ID_x^y's descendants (including itself).

- **Shared Key**: This algorithm is executed by a node (say, $ID_b^{L_1}$) in the hierarchy to generate a shared secret key with another node (say, $ID_a^{L_2}$) in the hierarchy.

  We know that each of the distributed secret keys corresponds to a pair of nodes in the hierarchy. Thus, these keys can be used directly as the shared-secret key for the corresponding pair. With input the key-ring of $ID_b^{L_1}$ as $\mathcal{R}_b^{L_1}$ and the identity $ID_a^{L_2}$, this algorithm outputs a secret key from $\mathcal{R}_b^{L_1}$ which corresponds to the pair $(ID_b^{L_1}, ID_a^{L_2})$. Note that there are no additional computations required in this algorithm.

3.5.2 Limitations in the Trivial Solution

It seems that the Trivial solution is a better choice than the HHKAS for resource constraint networks which require full resilience and inter-level key exchange. Trivial solution also provides the four functional properties presented in [16] and can be shown secure without any involvement of hardness assumptions. However, there are also certain limitations in the Trivial solution which are needed to be resolved for the real life applications -

1. From the protocol description, it is clear that the Trivial solution is not flexible towards a direct node admission i.e., a parent can not generate valid secret-keys for a new child without the help of its ancestors (including the PKG). Since in real life applications, especially in military, it is difficult for all nodes to come together and distribute the new keys whenever a new node gets admitted into the network. Therefore, in such applications, flexibility of a key exchange protocol towards a direct node admission is one of the must requirements.

2. In Trivial solution, all secret keys are generated only by the PKG. That means, an intermediate node can not generate the keys for any of its descendants and therefore all the keys of a lower level node are required to be transferred from PKG to it through all of its ancestors. To exemplify, with Trivial solution in an ℓ-levelled hierarchy, providing a secret key set of n elements to a leaf node will require around ℓn elements to be transferred throughout the hierarchy. This

"redundant" increase in the communication reflects in the performance and power consumption of MANETs. Note that these communications should not be considered as a "one-time cost" because these networks are exposed to cyber attacks and therefore key-refreshments are needed to be done periodically. This drawback is a good motivation for working towards H-ID-NIKE protocols with better hierarchical key delegation ways.

3. In Trivial solution, the root node is required to generate and store $\frac{N^2}{2}(1-\frac{1}{N})$ number of keys in its storage where N represents the total number of nodes in the network. The storage of root here is $\mathcal{O}(N^2)$ which is quite smaller in size when compared with the corresponding storage requirements in fully resilient HHKAS protocol. However, this storage requirement also increases polynomially with N. Therefore, for moderate N values, this protocol imposes a high workload on the root node (PKG). This drawback explains the necessity of an H-ID-NIKE protocol with key storage which is independent (or less dependant) from N.

3.6 BIOS-SOK Key Exchange Protocol

BIOS-SOK Key Exchange Protocol was introduced in 2017 by Tiwari [27] as a possible solution for the H-ID-NIKE problem. BIOS-SOK is a hybrid of two ID-NIKE protocols named as BIOS and SOK (both are discussed in Section 2.6). BIOS-SOK is a hierarchical, identity-based and non-interactive protocol which is shown to be secure under the M-H-IND-SK model [27].

3.6.1 Construction of BIOS-SOK

We can define BIOS-SOK using the following algorithms -

1. **Setup** : This algorithm is executed by the root node. With input the number of nodes n in level 1 and the length of the hierarchy ℓ, this algorithm runs the setup algorithms of both BIOS and SOK protocols. The algorithm then outputs msk as a key pool $\{K_0, K_1, \ldots, K_{n-1}\}$ and mpk as $(\mathbb{G}, \mathbb{G}_T, e, q, n, H, H_\theta, h)$, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order q, e is a bilinear pairing defined as $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and H, H_θ, h are hash functions defined as -

$$H : \{0,1\}^* \to \mathbb{G}, \qquad H_\theta : \mathbb{G}_T \to \mathbb{Z}_q^*, \qquad h : \{0,1\}^* \to \mathbb{Z}_q^*.$$

2. (a) **Key Delegation at level 1 ($\mathcal{D}_1$)** : This algorithm works same as key distribution algorithm of BIOS. Using this algorithm, each node in level 1 gets its secret key from the root node.

   (b) **Key Delegation at level $L \neq 1$ ($\mathcal{D}_L$)** : This algorithm is executed by a node at level $L-1$ (say, ID_m^{L-1}) to compute the secret keys for its child node at level L (say, ID_j^L). With input the secret key set of ID_m^{L-1} as $SK_{m(L-1)}$ and the child identity ID_j^L, this algorithm computes the tuple of ID_m^{L-1}'s shared secret keys with all the peers at level $L-1$ including itself. All these shared keys belong to either $\mathbb{G}_T$ or $\mathbb{Z}_q^*$ (depending upon the type of parent's shared secret keys). At this point, the algorithm maps these shared secret keys to $\mathbb{Z}_q^*$ using the hash function H_θ. Let us represent this mapped set by $SSK(ID_m^{L-1}) = \{d_1, \ldots, d_{n_{L-1}}\}$. Now, the algorithm outputs the secret key set for the child node ID_j^L from $SSK(ID_m^{L-1})$ as -

$$SK_{jL} = \{d_1 \cdot H(ID_j^L), \ldots, d_{n_{L-1}} \cdot H(ID_j^L)\} \text{ where } d_i \in \mathbb{Z}_q^*$$

3. (a) **Shared Key Computation between nodes at level 1 ($\mathcal{SHK}_1$)** : This algorithm works same as the Shared Key Computation algorithm of BIOS. Using this algorithm, each node in level 1 can compute a shared secret key with any other node at level 1.

   (b) **Shared Key Computation between nodes at level $L \neq 1$ ($\mathcal{SHK}_L$)** : This algorithm works same as the Shared Key Computation algorithm of SOK. We know that the secret key components of nodes beyond level 1 are elements from the group $\mathbb{G}$. That's why it is easier to see that the algorithm can use bilinear pairing (as in SOK) to compute a common shared key between the nodes at level $L > 1$.

   (c) **Shared Key Computation between different level nodes** : This algorithm is also derived from the Shared Key Computation algorithm of SOK protocol. To understand this algorithm, let us consider two nodes $ID_b^{L_1}$ at level L_1 and $ID_a^{L_2}$ at level L_2 who want to compute a shared secret key between them.

      i. **At upper level node $ID_b^{L_1}$'s end**:
      The algorithm computes the shared secret key between $ID_b^{L_1}$ and the ancestor of $ID_a^{L_2}$ at level L_1. Then using this shared secret key it computes the shared secret key between $ID_b^{L_1}$'s left most descendant at level $L_1 + 1$ and the ancestor of $ID_a^{L_2}$ at level $L_1 + 1$. This continues till the algorithm computes a shared secret key x between $ID_b^{L_1}$'s left most descendant

at level $L_2 - 1$ and the parent of $ID_a^{L_2}$. Note that this algorithm implicitly calls the corresponding $\mathcal{SHK}_L$ algorithm to do these same levelled shared key computations. Now, the algorithm outputs the shared secret key between $ID_b^{L_1}$ and $ID_a^{L_2}$ using x as -

$$\mathcal{SHK}(ID_b^{L_1}, ID_a^{L_2}) = e(H(ID_b^{L_1}), H(ID_a^{L_2}))^x.$$

ii. **At lower level node ID_j's end:**
The algorithm can directly compute $\mathcal{SHK}(ID_a^{L_2}, ID_b^{L_1})$ for the lower level node $ID_a^{L_2}$. This is because the key delegation in BIOS-SOK (as shown above) is done in such a way that $ID_a^{L_2}$ contains $x \cdot H(ID_a^{L_2})$ as a secret key in its secret key set SK_{aL_2}. Thus the algorithm computes and outputs -

$$\mathcal{SHK}(ID_a^{L_2}, ID_b^{L_1}) = e(x \cdot H(ID_a^{L_2}), H(ID_b^{L_1})).$$

Correctness: The correctness of the shared secret keys can be argued using the bilinear property of the pairing e i.e.,

$$\begin{aligned}\mathcal{SHK}(ID_a^{L_2}, ID_b^{L_1}) &= e(x \cdot H(ID_a^{L_2}), H(ID_b^{L_1})) \\ &= e(H(ID_a^{L_2}), H(ID_b^{L_1}))^x = e(H(ID_b^{L_1}), H(ID_a^{L_2}))^x \\ &= \mathcal{SHK}(ID_b^{L_1}, ID_a^{L_2})\end{aligned}$$

3.6.2 Limitations in BIOS-SOK

It is easy to notice that the key storage of a leaf node in BIOS-SOK depends linearly upon the number of nodes in the network ($K_{leaf} = \mathcal{O}(N)$) which makes it a better choice than the HHKAS or the Trivial solution in terms of key set size. However, there are some restrictions in the BIOS-SOK protocol which are not expected from an ideal H-ID-NIKE in the real life applications -

1. BIOS-SOK is claimed to provide the four functional properties presented in [16]. However, the full resilience property is defined over a restricted model. Which implies that in H-IND-SK model, BIOS-SOK is not fully resilient. In fact, in H-IND-SK model, BIOS-SOK is not resilient against a single compromisation from the set of left most descendants of the upper level target node up to the parent level of the lower levelled target node. In other words, for a hierarchy with N number of non-leaf nodes there are $(N - n_{\ell-1})$ nodes in the network

who when compromised, can leak shared secret keys of at least one of the "valid" target pairs (as defined in H-IND-SK model). Here $n_{\ell-1}$ denotes the number of nodes in the last non-leaf level.

2. The underlying model of BIOS-SOK (M-H-IND-SK) does not allow the adversary to make "Reveal" query on any of the ancestor pairs of the target nodes. Therefore this model does not capture one of the basic security requirements which is an adversary, even after compromising the shared secret keys for some ancestor pairs of the target nodes, should not be able to compute the shared secret key between the target nodes.

3. The key storage of a node in BIOS-SOK depends exponentially upon the hierarchical depth and polynomailly upon the branching factors up to level $\ell - 2$ in the hierarchy. Thus BIOS-SOK can only be a suitable choice for hierarchies with moderate hierarchical depths and moderate branching factors up to level $\ell - 2$.

4. BIOS-SOK supports both intra-level and inter-level shared key computations. However, in BIOS-SOK, two nodes $ID_b^{L_1}$ and $ID_a^{L_2}$ at different levels ($L_1 > L_2$) can compute a shared secret key between them if and only if there exists a set of left most descendants of $ID_b^{L_1}$ containing at least one node from each level in $(L_1 + 1)$ to $(L_2 - 1)$. This condition limits the choices and varieties of the hierarchy. Due to this restriction, BIOS-SOK is not a good choice for networks which require extreme flexibility as well as inter-level key exchange.

Chapter 4

α-BSOK Key Exchange Protocol

4.1 α-BSOK

In this section, we present α-BSOK key exchange protocol as a hybrid of BIOS and SOK protocols. We claim that α-BSOK is a possible solution for the open problem posed by Gennaro *et al.* [16] and it possesses the following properties:

1. *Fully Resilient:* The protocol is fully resilient against compromise of arbitrary number of nodes at any level in the hierarchy.
2. *Identity-based :* A node requires its own secret key and the other node's identity to compute a shared secret key between them.
3. *Non-Interactive:* Any pair of nodes can compute their shared secret key without any interaction.
4. *Decentralized or Hierarchical:* Any node in the hierarchy can generate secret keys for its descendants.

4.1.1 Protocol description of α-BSOK

For the description below, we are considering an ℓ levelled hierarchy with n nodes at level 1 and maximum branching factor b_{max} from level 1 to $\ell - 2$. In other words, all nodes in level 1 to $\ell - 2$ can have at most b_{max} children whereas PKG has n children. On the other hand, nodes at level $\ell - 1$ have no branching restrictions i.e. they can have arbitrarily large number of children (leaf nodes).

Similar to BIOS-SOK, α-BSOK protocol uses BIOS at level 1 and SOK afterwards. We provide the construction of α-BSOK protocol using the protocol description of BIOS and SOK. We can define α-BSOK protocol using the following algorithms:

- **Setup:** This algorithm is executed by the PKG.

  With input security parameter 1^λ, number of nodes in level 1 (n) and hierarchy depth (ℓ), this algorithm runs the Setup algorithms of BIOS and SOK protocols. The algorithm outputs master secret key msk as a key pool $\{K_0, K_1, \ldots, K_{n-1}\}$ from $\mathbb{Z}_q^*$ and public parameters mpk as $(\mathbb{G}, \mathbb{G}_T, e, q, n, h, H_1, H_2, H_3)$, where -

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T,\ h : \{0,1\}^* \to \mathbb{Z}_q^*,$$

$$H_1 : \{0,1\}^* \to \mathbb{G},\ H_2 : \{0,1\}^* \to \mathbb{Z}_q^* \text{ and } H_3 : \{0,1\}^* \to \mathbb{Z}_q^*.$$

- **Key Delegation at Level 1 ($\mathcal{D}_1$):** This algorithm is executed by the PKG.

  Let us denote the identity of a j^{th} node in level 1 as ID_j^1. With msk and a user's identity ID_j^1 as input, this algorithm runs the key-distribution algorithm of BIOS (as shown below in the Table 4.1) and outputs a set containing $\lceil \frac{n}{2} \rceil$ elements from $\mathbb{Z}_q^*$. We define this set as the secret key set $\mathcal{K}_{1j}$.

Nodes (ID_j^1)	Secret-Key sets ($\mathcal{K}_{1j}$)					
ID_0^1	K_0	$h(K_{\lceil \frac{n}{2} \rceil} \vert\vert ID_0^1)$	$h(K_{\lceil \frac{n}{2} \rceil+1} \vert\vert ID_0^1)$	$\cdots$	$h(K_{n-2} \vert\vert ID_0^1)$	$h(K_{n-1} \vert\vert ID_0^1)$
ID_1^1	K_1	$h(K_{\lceil \frac{n}{2} \rceil+1} \vert\vert ID_1^1)$	$h(K_{\lceil \frac{n}{2} \rceil+2} \vert\vert ID_1^1)$	$\cdots$	$h(K_{n-1} \vert\vert ID_1^1)$	$h(K_0 \vert\vert ID_1^1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ID_{n-1}^1	K_{n-1}	$h(K_{\lceil \frac{n}{2} \rceil-1} \vert\vert ID_{n-1}^1)$	$h(K_{\lceil \frac{n}{2} \rceil} \vert\vert ID_{n-1}^1)$	$\cdots$	$h(K_{n-3} \vert\vert ID_{n-1}^1)$	$h(K_{n-2} \vert\vert ID_{n-1}^1)$

Table 4.1 Key delegation at level 1 in α-BSOK protocol

- **Shared Key Computation at level 1 ($\mathcal{SHK}_1$):** This algorithm works exactly as Shared Key algorithm of BIOS over hash function h. Using this algorithm, a node ID_a^1 at level 1 first computes the BIOS shared secret key $\mathcal{SHK}'_1(ID_a^1, ID_b^1)$ with any other node ID_b^1 at level 1. Then the final shared key between these nodes is defined using H_3 hash function:

$$\mathcal{SHK}'_1(ID_a^1, ID_b^1) = \begin{cases} h(K_a || ID_b^1) \text{ if } (b-a) \mod n \leq \lfloor \frac{n}{2} \rfloor, \\ h(K_b || ID_a^1) \text{ if } (b-a) \mod n > \lfloor \frac{n}{2} \rfloor. \end{cases}$$

$$\mathcal{SHK}_1(ID_a^1, ID_b^1) = H_3\Big[\mathcal{SHK}'_1(ID_a^1, ID_b^1)\Big].$$

Note that $\mathcal{SHK}'_1$ is used in key delegation for any lower level descendant. Correctness of this shared key algorithm comes trivially from the correctness of BIOS Shared Key algorithm.

- **Key Delegation at level L ($\mathcal{D}_L$, $2 \leq L \leq \ell$):** This algorithm is executed by a node in the level $L-1$ to generate keys for its children in level L. Let us assume m^{th} node in level $L-1$ is the parent of the j^{th} node in level L and n_i represents total number of nodes from first level to the i^{th} level of the hierarchy ($n_0 = 0, n_1 = n$). At first, with $\mathcal{K}_{(L-1)m}$, ID_m^{L-1} and ID_j^L as input, this algorithm computes involved parent node ID_m^{L-1}'s shared secret key set say $\{d_1, \ldots, d_{n_{L-1}}\}$ i.e., shared secret keys of ID_m^{L-1} with all the peers up to level $L-1$ including itself. Here d_i is defined as -

$$d_i = \begin{cases} \mathcal{SHK}'_{r(L-1)}(ID_m^{L-1}, ID_i^r) \text{ if } i \in (n_{r-1}, n_r] \text{ and } 1 \leq r \neq L-1, \\ \mathcal{SHK}'_{L-1}(ID_m^{L-1}, ID_i^{L-1}) \text{ if } i \in (n_{L-2}, n_{L-1}]. \end{cases}$$

  All these shared keys belong to either $\mathbb{G}_T$ or $\mathbb{Z}_q^*$ (depending upon the type of parent's shared secret keys). At this point, the algorithm maps these shared secret keys to $\mathbb{Z}_q^*$ using the hash function H_2. Once it gets the set of these mapped shared secret keys, it multiplies them to $Q_{ID_j^L}$ which is the H_1 hash of its child's identity ID_j^L. The resultant set $\{H_2(d_1) \cdot Q_{ID_j^L}, \ldots, H_2(d_{n_{L-1}}) \cdot Q_{ID_j^L}\}$ is provided to the child as the secret key set $\mathcal{K}_{Lj}$. This secret key set contains n_{L-1} elements of group $\mathbb{G}$.

- **Shared Key Computation at level L ($\mathcal{SHK}_L$, $2 \leq L \leq \ell$):** This algorithm is used to compute shared secret key between a pair consisting nodes of level $L \geq 2$. This algorithm uses Shared Key algorithm of SOK protocol. (We know that the secret keys of nodes for shared-key generation beyond level 1 are elements of $\mathbb{G}$ and therefore this algorithm can use bilinear pairing to compute such shared secret keys.)

  Suppose two nodes at level L, say ID_a^L and ID_b^L want to compute a shared secret key.

  At ID_a^L's end:

  ID_a^L checks its secret key set $\mathcal{K}_{La}$ for the group element it received by the $\mathcal{D}_L$ algorithm as a multiplication of their parent's shared secret key's H_2 hash with the H_1 hash of its identity i.e.,

$$H_2[\mathcal{SHK}'_{L-1}(parent(ID_a^L), parent(ID_b^L))] \cdot H_1(ID_a^L).$$

Now, the shared secret key is computed as -

$$\mathcal{SHK}'_L(ID_a^L, ID_b^L) = e(x_1 \cdot H_1(ID_a^L), H_1(ID_b^L))$$

$$\text{where } x_1 = H_2[\mathcal{SHK}'_{L-1}(parent(ID_a^L), parent(ID_b^L))],$$

$$\mathcal{SHK}_L(ID_a^L, ID_b^L) = H_3\Big[\mathcal{SHK}'_L(ID_a^L, ID_b^L)\Big].$$

At ID_b^L's end:

ID_b^L checks its secret key set $\mathcal{K}_{Lb}$ for the group element it received by the $\mathcal{D}_L$ algorithm as a multiplication of their parent's shared secret key's H_2 hash with the H_1 hash of its identity i.e.,

$$H_2[\mathcal{SHK}'_{L-1}(parent(ID_b^L), parent(ID_a^L))] \cdot H_1(ID_b^L).$$

Now, the shared secret key is computed as -

$$\mathcal{SHK}'_L(ID_b^L, ID_a^L) = e(x_2 \cdot H_1(ID_b^L), H_1(ID_a^L))$$

$$\text{where } x_2 = H_2[\mathcal{SHK}'_{L-1}(parent(ID_b^L), parent(ID_a^L))],$$

$$\mathcal{SHK}_L(ID_b^L, ID_a^L) = H_3\Big[\mathcal{SHK}'_L(ID_b^L, ID_a^L)\Big].$$

Correctness: We have iterative correctness from $\mathcal{SHK}_1$ algorithm to $\mathcal{SHK}_{L-1}$. Hence, we have, $x_1 = x_2$ and therefore using the bilinear property of the pairing e, we have -

$$\mathcal{SHK}_L(ID_a^L, ID_b^L) = \mathcal{SHK}_L(ID_b^L, ID_a^L).$$

Note that $\mathcal{SHK}'_L$ is used as shared secret in key delegation for any lower level descendant.

- **Shared Key Computation between levels L_1 and L_2 ($\mathcal{SHK}_{L_1L_2}$):** This algorithm is used to compute shared secret key between a pair consisting one node from level L_1 and another from level L_2 where $1 \leq L_1 < L_2 \leq \ell$. We know that the secret keys of nodes for shared-key generation beyond level 1 are elements of $\mathbb{G}$ and therefore this algorithm can use bilinear pairing to compute such shared secret keys.

  Suppose two nodes, say $ID_a^{L_2}$ and $ID_b^{L_1}$ want to compute a shared secret key.

At $ID_b^{L_1}$'s end: Since $ID_b^{L_1}$ can generate $\mathcal{SHK}'_{L_1}(ID_b^{L_1}, ancestor^{L_1}(ID_a^{L_2}))$, its shared secret key with $ID_a^{L_2}$ can be computed as -

$$\begin{aligned}
x_{L_1} &= H_2[\mathcal{SHK}'_{L_1}(ID_b^{L_1}, ancestor^{L_1}(ID_a^{L_2}))] \cdot H_1(ID_b^{L_1}) \\
x_{L_1+1} &= H_2[e(x_{L_1}, H_1(ancestor^{L_1+1}(ID_a^{L_2})))] \cdot H_1(ID_b^{L_1}) \\
. & \qquad\qquad . \\
. & \qquad\qquad . \\
x_{L_2-1} &= H_2[e(x_{L_2-2}, H_1(ancestor^{L_2-1}(ID_a^{L_2})))] \cdot H_1(ID_b^{L_1})
\end{aligned}$$

$$\begin{aligned}
\mathcal{SHK}'_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) &= e(x_{L_2-1}, H_1(ID_a^{L_2})) \\
\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) &= H_3\Big[\mathcal{SHK}'_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})\Big].
\end{aligned}$$

Here $ancestor^i(X)$ represents i^{th} levelled ancestor of the identity X.

At $ID_a^{L_2}$'s end: $ID_a^{L_2}$ checks its secret key set $\mathcal{K}_{L_2a}$ for the group element it received by the $\mathcal{D}_{L_2}$ algorithm as a multiplication of $ID_b^{L_1}$ and its own parent's shared secret key's H_2 hash with the H_1 hash of its identity i.e.,

$$H_2[\mathcal{SHK}'_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})] \cdot H_1(ID_a^{L_2}).$$

Now, the shared secret key is computed as -

$$\mathcal{SHK}'_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1}) = e(x_1 \cdot H_1(ID_a^{L_2}), H_1(ID_b^{L_1}))$$

$$\text{where } x_1 = H_2[\mathcal{SHK}'_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})],$$

$$\mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1}) = H_3\Big[\mathcal{SHK}'_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})\Big]$$

$$\Bigg\{ \text{For } L_1 = L_2 - 1, \text{ we define } \mathcal{SHK}'_{L_1(L_2-1)}(X,Y) = \mathcal{SHK}'_{L_1}(X,Y) \Bigg\}.$$

Correctness: In the above computation, we can notice that at ID_a's end, the corresponding stored secret key was computed by its parent in a similar manner to ID_b's computation of x_{L_2-1}. Therefore, from the correctness argument of the

$\mathcal{SHK}_{L_1}$, we have -

$$x_{L_2-1} = x_1 \cdot H_1(ID_b^{L_1})$$
$$\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) = \mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1}).$$

Note that $\mathcal{SHK}'_{L_1L_2}$ is used as shared secret in key delegation for any lower level descendant.

Remark: In the α-BSOK protocol, we have used four hash functions and three of them have same image sets. We can replace these three hash functions (h, H_2, H_3) by a single hash function $H : \{0,1\}^* \to \mathbb{Z}_q^*$ as described below:

$$\begin{aligned} h(x) &= H(0||x) \; \forall \; x \in \{0,1\}^*, \\ H_2(y) &= H(1||y) \; \forall \; y \in \mathbb{G}_T, \\ H_3(z) &= H(2||z) \; \forall \; z \in \mathbb{Z}_q^* \text{ or } \mathbb{G}_T. \end{aligned}$$

We can notice that in this protocol, any node can do the shared key computation with any other node in the network. We can also see that for different level nodes' shared key computation, higher level node requires more pairing computations than the lower level node. One way to eliminate these computations is by storing these keys in the higher level node's storage.

Trading Space with Time: Here we describe an optional algorithm to generate and store these additional keys for faster shared key computations.

- **Supplementary Key Generation at level L ($\mathcal{D}_L^*$, $1 \leq L < \ell$):** This algorithm is executed by a node in the level L. Let us assume that the set of additional keys for a node ID_j^L is denoted by $\mathcal{K}'_{Lj}$ then we define $\mathcal{K}'_{Lj}$ as -

$$\mathcal{K}'_{Lj} = \bigcup_{i=L}^{\ell-1} \bigcup_{k=n_{i-1}}^{n_i} \{H_2[\mathcal{SHK}'(ID_j^L, ID_k^i)] \cdot Q_{ID_j^L}\}.$$

Now the total number of keys in ID_j^L's key space is -

$$|\mathcal{K}_{Lj}| + |\mathcal{K}'_{Lj}| = n_{L-1} + (n_{\ell-1} - n_{L-1}) = n_{\ell-1}.$$

For security proof and further analysis, we have used α-BSOK with this traded space over time.

4.1.2 Security of α-BSOK

We argue the security for α-BSOK protocol in the H-IND-SK model.

Security Proof

Since α-BSOK is a hybrid of two different protocols, we show the proof of security using the following two cases:

- **Case 1:** When both the target nodes are at level 1: We provide the proof for this case using the security of BIOS, which is the underlying protocol at level 1.
- **Case 2:** When at least one of the target nodes lies beyond level 1: We provide the proof for this case using the S-DBDH assumption.

It is easy to notice that these two cases contain all possible scenarios. As according to these two cases, we consider two different types of adversaries against α-BSOK protocol.

- **Case 1:** (Both the target nodes are at level 1) Let us describe the essential restrictions on the adversary to prevent it from winning the game trivially in Case 1:
  1. Adversary is not allowed to compromise any of the target nodes.
  2. Adversary is not allowed to ask for the shared secret key between the target identities.

We show how to construct an efficient adversary $\mathcal{B}$ against BIOS, using an H-IND-SK adversary $\mathcal{A}_1$ which has a non-negligible advantage against α-BSOK protocol in Case 1. The game between the BIOS challenger $\mathcal{C}$ and its adversary $\mathcal{B}$ starts with $\mathcal{C}$ first providing $\mathcal{B}$ the public parameters for BIOS (mpk_{BIOS}). $\mathcal{C}$ also computes a key-ring for BIOS (msk_{BIOS}) and keeps it to itself. $\mathcal{C}$ controls h as random oracle. Note that $\mathcal{B}$ does not have any control over $\mathcal{C}$. $\mathcal{B}$ only seeks the help of $\mathcal{C}$ to respond the queries which are related to BIOS. Now, $\mathcal{B}$ who is the challenger for α-BSOK protocol interacts with $\mathcal{A}_1$ in the following manner:

- **Setup:** $\mathcal{B}$ gives $mpk = (\mathbb{G}, \mathbb{G}_T, e, P, h, H_1, H_2, H_3)$ to $\mathcal{A}_1$, where H_2 and H_3 are random oracles controlled by $\mathcal{B}$. $\mathcal{A}_1$ can make following queries:
- **h-queries:** $\mathcal{B}$ receives queries of the form $x_{ij} = (K_i||ID_{1j})$ from $\mathcal{A}_1$. It forwards such queries to $\mathcal{C}$. $\mathcal{C}$ responds back the $h(x_{ij})$ to $\mathcal{B}$. $\mathcal{B}$ then forwards this response to $\mathcal{A}_1$.

- **H_1-queries:** $\mathcal{B}$ provides the description of this cryptographic hash function to the adversary $\mathcal{A}_1$. With input an identity ID, this function outputs an element from $\mathbb{G}$.

- **H_2-queries:** $\mathcal{B}$ receives an element γ from $\mathbb{G}_T$ as query. In response to this query, $\mathcal{B}$ maintains an H_2-table with entries of the form $(\gamma, H_2(\gamma))$. If the queried argument is already there in H_2-table, the corresponding hash value is given to $\mathcal{A}_1$ as response. Else, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_2-table and the chosen element is provided to $\mathcal{A}_1$ as response.

- **H_3-queries:** $\mathcal{B}$ receives an element κ either from $\mathbb{G}_T$ or from $\mathbb{Z}_q^*$ as query. In response to this query, $\mathcal{B}$ maintains an H_3-table with entries of the form $(\kappa, H_3(\kappa))$. If the queried argument is already there in H_3-table, the corresponding hash value is given to $\mathcal{A}_1$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_3-table and the chosen element is provided to $\mathcal{A}_1$ as response.

- **Extract queries:** $\mathcal{A}_1$ provides an identity ID_i to $\mathcal{B}$. $\mathcal{B}$ responds to this extract query in the following manner:

  1. If $ID_i \in \mathcal{ID}^1$ then $\mathcal{B}$ forwards ID_i to $\mathcal{C}$ who then responds back with an appropriate secret key set K_i consisting $\frac{n_1+1}{2}$ elements from $\mathbb{Z}_q^*$. $\mathcal{B}$ generates another key set consisting $n_{\ell-1}$ elements from $\mathbb{G}$ either chosen uniformly at random or defined using previous queries. It sends this set along with the K_i to $\mathcal{A}_1$ as the secret key set of ID_i.
  2. If $ID_i \notin \mathcal{ID}^1$ then $\mathcal{B}$ responds with a key set consisting $n_{\ell-1}$ elements from $\mathbb{G}$ either chosen uniformly at random or defined using previous queries. For that, it may need to check relevant entries in the H_2 and H_3 hash tables. If the entries are not there, it may need to make relevant hash queries. $\mathcal{B}$ may also need to perform relevant h-queries to $\mathcal{C}$.

  All extract queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Reveal queries:** $\mathcal{A}_1$ provides an identity pair $\{ID_i, ID_j\}$. $\mathcal{B}$ responds to this reveal query in the following manner:

  1. If $\{ID_i, ID_j\} \subseteq \mathcal{ID}^1$ then $\mathcal{B}$ forwards the queried pair to $\mathcal{C}$ who then responds back with an appropriate shared secret key. $\mathcal{B}$ forwards this response to $\mathcal{A}_1$.

2. If $\{ID_i, ID_j\} \not\subseteq \mathcal{ID}^1$, then in order to respond this query, $\mathcal{B}$ may need to check relevant entries in the H_2 and H_3 hash tables. If the entries are not there, it may need to make relevant hash queries. $\mathcal{B}$ may also need to perform relevant h-queries to $\mathcal{C}$. $\mathcal{B}$ responds with an element from $\mathbb{Z}_q^*$ either chosen uniformly at random or defined using previous queries.

All reveal queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Test query:** $\mathcal{A}_1$ makes a single Test query on a pair of identities (say, ID_x and ID_y). Note that both target nodes are required to be in level 1 only. $\mathcal{B}$ forwards this pair to $\mathcal{C}$ as the target pair. $\mathcal{C}$ responds back with either the real shared secret key for this pair or a random element from $\mathbb{Z}_q^*$. $\mathcal{B}$ forwards the response of $\mathcal{C}$ to $\mathcal{A}_1$.

  In the end, $\mathcal{A}_1$ outputs a guess bit $\delta' \in \{0,1\}$. $\mathcal{B}$ forwards this guess bit as δ to $\mathcal{C}$.

Here, one can see that $\mathcal{B}$'s response is correct whenever $\mathcal{A}_1$'s response is correct, which means the advantage of $\mathcal{B}$ against BIOS is same as the advantage of $\mathcal{A}_1$ against the α-BSOK protocol.

Analysis : Formally, if the advantage of $\mathcal{A}_1$ against α-BSOK protocol is $Adv_{\alpha\text{-}BSOK}$, then the advantage Adv_{BIOS} for $\mathcal{B}$ against BIOS can be given as:

$$Adv_{BIOS} = Adv_{\alpha\text{-}BSOK}.$$

Thus, if $Adv_{\alpha\text{-}BSOK}$ is non-negligible, then we can see that Adv_{BIOS} is also non-negligible which contradicts the assumption of DCSK problem (see Section 2.6.2). Hence, $Adv_{\alpha\text{-}BSOK}$ must be negligible.

- **Case 2:** (When at least one of the target nodes lies beyond level 1; let the lower levelled target node lies at level m in the hierarchy; $1 < m \leq \ell$) We define the restrictions over the adversary so that it does not win the security game trivially in Case 2 as follows:

  1. Adversary is not allowed to compromise the target nodes.
  2. Adversary is not allowed to compromise any of the ancestors of the target nodes.

3. Adversary is not allowed to ask for the shared secret key between the target nodes.

We show how to construct an efficient S-DBDH adversary $\mathcal{B}$, using an H-IND-SK adversary $\mathcal{A}_2$ which has a non-negligible advantage against α-BSOK protocol in Case 2. The game between the S-DBDH challenger $\mathcal{C}$ and its adversary $\mathcal{B}$ starts with $\mathcal{C}$ first providing $\mathcal{B}$ the S-DBDH instance (aP, cP, Q) with $(\mathbb{G}, \mathbb{G}_T, e, q, P)$. Note that $\mathcal{B}$ does not have any control over $\mathcal{C}$. $\mathcal{B}$ only seeks the help of $\mathcal{C}$ to generate the instance of the hard problem. Now, $\mathcal{B}$ who is the challenger for α-BSOK protocol interacts with $\mathcal{A}_2$ in the following manner:

- **Setup:** $\mathcal{B}$ gives $mpk = (\mathbb{G}, \mathbb{G}_T, e, P, h, H_1, H_2, H_3)$ to $\mathcal{A}_2$, where h, H_1, H_2 and H_3 are random oracles controlled by $\mathcal{B}$. Let $q_{H_{1m}}$ and $q_{H_{1\bar{m}}}$ be the number of H_1 hash queries done by $\mathcal{A}_2$ in the level m and in the levels from 1 to m respectively during the simulation. $\mathcal{B}$ chooses two indices I and J uniformly at random from $\{1, 2, \ldots, q_{H_{1m}}\}$ and $\{1, 2, \ldots, q_{H_{1\bar{m}}}\}$ respectively. $\mathcal{A}_2$ can make the following queries:

- **h-queries:** $\mathcal{B}$ receives queries of the form $x_{ij} = (K_i || ID_{1j})$ from $\mathcal{A}_2$. In response to this query, $\mathcal{B}$ maintains an h-table with entries of the form $(x_{ij}, h(x_{ij}))$. If the queried argument is already there in h-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in h-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **$\mathbf{H_1}$-queries:** $\mathcal{A}_2$ provides an identity ID from the identity space. To Respond such queries $\mathcal{B}$ maintains an H_1-table with entries $(ID, r, t, H_1(ID))$. If the provided ID is already there in the table, $\mathcal{B}$ responds with corresponding $H_1(ID)$ entry. Otherwise i-th query by $\mathcal{A}_2$ is responded in the following manner:

  1. if $i = I$, then for an integer $s \in_R \mathbb{Z}_q^*$, $\mathcal{B}$ adds entries $(ID_I, s, \perp, saP)$ and $(ID_J, s^{-1}, \perp, s^{-1}aP)$ to the table and returns $H_1(ID_I) = saP$.
  2. if $i = J$, then for an integer $s \in_R \mathbb{Z}_q^*$, $\mathcal{B}$ adds entries $(ID_I, s, \perp, saP)$ and $(ID_J, s^{-1}, \perp, s^{-1}aP)$ to the table and returns $H_1(ID_J) = s^{-1}aP$.
  3. if ID_i is an ancestor of either ID_I or ID_J at the $(L \neq 1)^{th}$ level but not for both then there exists some ID_j in the level of ID_i who is corresponding ancestor of the other target node. For this query, $\mathcal{B}$ chooses integers $t_L, r_{L-1} \in_R \mathbb{Z}_q^*$ and adds entries $(ID_i, r_{L-1}, t_L, r_{L-1}t_L aP)$ and $(ID_j, r_{L-1}, t_L^{-1}, r_{L-1}t_L^{-1}aP)$ to the table and returns $H_1(ID_i) = r_{L-1}t_L aP$.

4. if ID_i is a common ancestor of ID_I and ID_J at the $(L \neq 1)^{th}$ level then, for this query, $\mathcal{B}$ chooses integer $r_{L-1} \in_R \mathbb{Z}_q^*$, adds an entry $(ID_i, r_{L-1}, \perp, r_{L-1}aP)$ to the table and returns $H_1(ID_i) = r_{L-1}aP$.
5. Otherwise $\mathcal{B}$ chooses s_i uniformly at random from $\mathbb{Z}_q^*$, adds $(ID_i, s_i, \perp, s_iP)$ to the H_1-table, and returns $H_1(ID_i) = s_iP$.

- **H$_2$-queries:** $\mathcal{B}$ receives an element γ from $\mathbb{G}_T$ as query. In response to this query, $\mathcal{B}$ maintains an H_2-table with entries of the form $(\gamma, H_2(\gamma))$. If the queried argument is already there in H_2-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Else, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_2-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **H$_3$-queries:** $\mathcal{B}$ receives an element κ either from $\mathbb{G}_T$ or from $\mathbb{Z}_q^*$ as query. In response to this query, $\mathcal{B}$ maintains an H_3-table with entries of the form $(\kappa, H_3(\kappa))$. If the queried argument is already there in H_3-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_3-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **Extract queries:** $\mathcal{A}_2$ provides an identity ID_i. $\mathcal{B}$ makes an H_1 hash query on ID_i, if this has not been queried already. $\mathcal{B}$ responds to the extract query in the following manner:

  1. If $ID_i \in \{ID_I, ID_J\}$ or is an ancestor of ID_I or ID_J then $\mathcal{B}$ aborts the simulation.
  2. If ID_i is a sibling of either ID_I or ID_J then $\mathcal{B}$ first finds an entry $(ID_i, s_i, \perp, H_1(ID_i))$ in the H_1-table. It defines one of the secret key component of ID_i which corresponds to the shared secret key between the parent pair of ID_I and ID_J as $k = s_i(cP)$. For remaining keys, $\mathcal{B}$ may need to check relevant entries in the four hash tables (h, H_1, H_2 and H_3-table). If the entries are not there, it may need to make relevant hash queries. Finally, $\mathcal{B}$ responds with a set of $n_{\ell-1}$ elements (including k) from $\mathbb{G}$.
  3. If ID_i is a sibling of either target identities' $(L \neq 1)^{th}$ level ancestor then $\mathcal{B}$ first finds an entry $(ID_i, s_i, \perp, H_1(ID_i))$ in the H_1-table. It defines one of the secret key component of ID_i which corresponds to the shared secret key between the $(L-1)^{th}$ level ancestors of ID_I and ID_J as $k = s_i r_{L-1}^{-2}(cP)$. For remaining keys, $\mathcal{B}$ may need to check relevant entries in the four hash tables. If the entries are not there, it may need to make relevant hash queries. Finally, $\mathcal{B}$ responds with a set of $n_{\ell-1}$ elements (including k) from $\mathbb{G}$.

4. For remaining ID_is', $\mathcal{B}$ follows the α-BSOK protocol. For that, it may need to check relevant entries in the four hash tables. If the entries are not there, it may need to make relevant hash queries. $\mathcal{B}$ responds these queries with a set consisting elements from $\mathbb{Z}_q^*$ and $\mathbb{G}$. These elements are either chosen uniformly at random or defined using previous queries. Depending upon the position of queried node in the hierarchy, the number of elements in the response set is either $n_{\ell-1}$ or $\left(n_{\ell-1} + \frac{n_1+1}{2}\right)$.

All extract queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Reveal queries:** $\mathcal{A}_2$ provides an identity pair $\{ID_i, ID_j\}$. $\mathcal{B}$ makes H_1 hash queries on the identities ID_i and ID_j if this has not been already done. $\mathcal{B}$ responds to the reveal query in the following manner:

  1. If $\{ID_i, ID_j\} = \{ID_I, ID_J\}$, then $\mathcal{B}$ aborts the simulation.
  2. If both ID_i and ID_j are ancestors of ID_I and ID_J respectively at L^{th} level then $\mathcal{B}$ chooses an integer u_L uniformly at random from $\mathbb{Z}_q^*$ and responds with $\mathcal{SHK}_L(ID_i, ID_j) = u_L$.
  3. For $L \neq 1$, if the identity ID_i is a sibling of ID_I's L^{th} level ancestor and ID_j is either ID_J's L^{th} level ancestor or a sibling of ID_J's L^{th} level ancestor then $\mathcal{B}$ first obtains the H_1 hash query $(ID_i, s_i, \perp, s_iP)$. It then responds with $\mathcal{SHK}_L(ID_i, ID_j) = H_3[e(s_i r_{L-1}^{-2}(cP), H_1(ID_j))]$.
  4. For $L \neq 1$, if the identity ID_i is a sibling of ID_J's L^{th} level ancestor and ID_j is either ID_I's L^{th} level ancestor or a sibling of ID_I's L^{th} level ancestor then $\mathcal{B}$ first obtains the H_1 hash query $(ID_i, s_i, \perp, s_iP)$. It then responds with $\mathcal{SHK}_L(ID_i, ID_j) = H_3[e(s_i r_{L-1}^{-2}(cP), H_1(ID_j))]$.
  5. If the identity ID_i is a sibling of ID_I and ID_j is either ID_J or a sibling of ID_J then $\mathcal{B}$ first obtains the H_1 hash query $(ID_i, s_i, \perp, s_iP)$. It then responds with $\mathcal{SHK}_L(ID_i, ID_j) = H_3[e(s_i(cP), H_1(ID_j))]$.
  6. If the identity ID_i is a sibling of ID_J and ID_j is either ID_I or a sibling of ID_I then $\mathcal{B}$ first obtains the H_1 hash query $(ID_i, s_i, \perp, s_iP)$. It then responds with $\mathcal{SHK}_L(ID_i, ID_j) = H_3[e(s_i(cP), H_1(ID_j))]$.
  7. For all remaining pairs, $\mathcal{B}$ follows the α-BSOK protocol. For that, it may need to check relevant entries in the four hash tables. If the entries are not there, it may need to make relevant hash queries. $\mathcal{B}$ responds these queries with an element from $\mathbb{Z}_q^*$ either chosen uniformly at random or defined using previous queries.

All reveal queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Test query:** $\mathcal{A}_2$ makes a single Test query on a pair of identities. If $\mathcal{A}_2$ does not choose ID_I and ID_J as the challenge identities, then $\mathcal{B}$ aborts the simulation and fails. Otherwise $\mathcal{B}$ outputs $H_3(Q) \in \mathbb{Z}_q^*$ as the shared secret key for the challenge identities.

  In the end, $\mathcal{A}_2$ outputs a guess bit $\delta' \in \{0,1\}$.

Note that if $\delta = 1$ i.e., the S-DBDH instance really contained $e(P,P)^{a^2c}$ as Q, then -

$$\begin{aligned} H_3(Q) = H_3\Big(e\big(P,P\big)^{a^2c}\Big) = H_3\Big(e\big(aP, c(aP)\big)\Big) = H_3\Big(e\big(saP,\ c(s^{-1}aP)\big)\Big) \\ = H_3\Big(e\big(H(ID_I),\ SK_{ID_J}\big)\Big) = SHK(ID_I,\ ID_J). \end{aligned}$$

Also,

$$\begin{aligned} Q = e\big(P,P\big)^{a^2c} = e\big(aP, c(aP)\big) &= e\big(r_{L-1}t_L aP,\ r_{L-1}^{-2}c(r_{L-1}t_L^{-1}aP)\big) \\ &= e\big(H(Ancestor^L(ID_I)),\ SK_{Ancestor^L(ID_J)}\big) \\ &= \mathcal{SHK}'_L(Ancestor^L(ID_I),\ Ancestor^L(ID_J))\ \forall\ L \neq 1, \end{aligned}$$

here if $Ancestor^L(ID_I) = Ancestor^L(ID_J)$ then $t_L = t_L^{-1} = 1$.

In other words, here Q corresponds to the real shared secret key between the challenge identities as well as to the key delegation shared key ($\mathcal{SHK}'$) between any of their ancestor pairs.

In this case, if the adversary does not target an ancestor pair of the challenge identities to compute the shared secret between the challenge identities then to respond $\delta' = 1$ with non-negligible advantage, adversary has to query H_3 hash of Q which can be easily verified by the simulator.

However, if the adversary targets an ancestor pair of the challenge identities to compute the shared secret between the challenge identities then it has to query $H_2(Q)$. Note that H_2 is modeled as a random oracle here, therefore, to query $H_2(Q)$ adversary must be able to compute $Q = e(P,P)^{a^2c}$. If the adversary can compute $e(P,P)^{a^2c}$ with some non-negligible advantage and queries its hash during the simulation then the simulator can output $\delta' = 1$ to the S-DBDH challenger with the same advantage by verifying

$H_2(Q)$ with this hash query in the H_2-table.

On the other hand, if $Q = e(P,P)^r$, for some random $r \in \mathbb{Z}_q^*$, then the test query response by $\mathcal{B}$ is a random element from $\mathbb{Z}_q^*$ which corresponds to $\delta' = \delta = 0$.

In this game, we say an adversary wins if $H_2(Q)$ is queried or $\delta' = \delta$. Let $\mathcal{U}$ be the event that $\mathcal{B}$ does not abort during its simulation under H-IND-SK model. Since $q_{H_{1m}} \leq q_{H_{1\bar{m}}} \leq q_{H_1}$. We can easily see that -

$$Pr(\mathcal{U}) \leq \frac{1}{q_{H_1}^2}.$$

Analysis: To formally analyze the advantage of $\mathcal{A}_2$ against α-BSOK, let us first define some events:

A = An event when adversary wins the security game.

B = An event when adversary queries $H_2(Q)$.

C = An event when adversary solves the S-DBDH at leaf level.

One can easily see that $Pr(C) \geq Pr(B)$. For simplicity, let us assume, $m > 2$. Now, the probability that the adversary wins the security game can be defined as -

$$\begin{aligned}
Pr(A) &= Pr(A \cap B) + Pr(A \cap \bar{B}) \\
Pr(A) &= Pr(A \cap B) + Pr(A \cap \bar{B} \cap C) + Pr(A \cap \bar{B} \cap \bar{C}) \\
Pr(A) &= Pr(A|B) \cdot Pr(B) + Pr(A \cap \bar{B}|C) \cdot Pr(C) + Pr(A|\bar{B} \cap \bar{C}) \cdot Pr(\bar{B} \cap \bar{C}) \\
Pr(A) &\leq Pr(B) + Pr(C) + Pr(A|\bar{B} \cap \bar{C}) \\
Pr(A) &\leq 2Pr(C) + Pr(A|\bar{B} \cap \bar{C}) \\
Pr(A) &\leq \frac{2Adv_{\text{S-DBDH}}}{Pr(\mathcal{U})} + \frac{1}{2} \\
Pr(A) - \frac{1}{2} &\leq 2q_{H_1}^2 (Adv_{\text{S-DBDH}}) \\
Adv_{\alpha\text{-BSOK}} &\leq 2q_{H_1}^2 (Adv_{\text{S-DBDH}}).
\end{aligned}$$

Hence, we have,

$$\frac{1}{2q_{H_1}^2} Adv_{\alpha\text{-BSOK}} \leq Adv_{\text{S-DBDH}}.$$

Since q_{H_1} is polynomial in number, if $Adv_{\alpha\text{-BSOK}}$ is non-negligible, then we can see that $Adv_{\text{S-DBDH}}$ is also non-negligible which contradicts the hardness assumption of

S-DBDH problem. Hence, $Adv_{\alpha\text{-}BSOK}$ must be negligible. $\square$

Remark: Using Coron's technique [9] for the Case 2 gives the security reduction as -

$$\left(\frac{\ell}{eq_E}\right)^{2\ell} \cdot Adv_{\alpha\text{-}BSOK} \leq Adv_{S\text{-}DBDH}.$$

Here q_E denotes the number of extract or reveal queries done by the adversary during the simulation. We can write the final expression as -

$$max\left\{\left(\frac{\ell}{eq_E}\right)^{2\ell}, \frac{1}{2q_{H_1}^2}\right\} Adv_{\alpha\text{-}BSOK} \leq Adv_{S\text{-}DBDH}.$$

4.1.3 Key Escrow Problem

The problem of key escrow in ID-based protocols is known since the beginning of IBC and has been studied for many years. By definition, key escrow is an authorized party which can regenerate private keys for some other parties in the network. In α-BSOK, every intermediate node in the hierarchy is a key escrow because it can regenerate all private and shared keys used in the subtree of its descendants. This inherent key escrow property of α-BSOK is desirable in some scenarios, such as governmental and military applications (to monitor lower level nodes), but in many other applications this property is considered as a limitation. In next section, we present a tweak in α-BSOK to avoid the issue of key escrow for such applications.

4.2 β-BSOK : A variant of α-BSOK

In this section, we present a variant of α-BSOK key exchange protocol named as β-BSOK to avoid the key escrow problem and to achieve a tighter proof of security under the same H-IND-SK model.

4.2.1 Protocol description of β-BSOK

For the description below, we are considering an ℓ levelled hierarchy with n nodes at level 1 and maximum branching factor b_{max} from level 1 to $\ell - 2$. In other words, all nodes in level 1 to $\ell - 2$ can have at most b_{max} children whereas PKG has n children. On the other hand, nodes at level $\ell - 1$ have no branching restrictions i.e. they can have arbitrarily large number of children (leaf nodes).

We provide the construction of β-BSOK protocol using the protocol description of α-BSOK. We can define β-BSOK protocol using the following algorithms:

- **Setup:** This algorithm is executed by the PKG.

  With input security parameter 1^λ, number of nodes in level 1 (n) and hierarchy depth (ℓ), this algorithm runs the Setup algorithms of BIOS and SOK protocols. The algorithm outputs master secret key msk as two key pools $\{K_0, K_1, \ldots, K_{n-1}\}$ and $\{y_1, y_2, \ldots, y_\ell\}$ from $\mathbb{Z}_q^*$ and public parameters mpk as $(\mathbb{G}, \mathbb{G}_T, e, q, n, h, H_1, H_2, H_3)$, where -

  $$e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T,\ h : \{0,1\}^* \rightarrow \mathbb{Z}_q^*,$$

  $$H_1 : \{0,1\}^* \rightarrow \mathbb{G},\ H_2 : \mathbb{G}_T \rightarrow \mathbb{Z}_q^* \text{ and } H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*.$$

  For the following algorithms, let us define a term called "fertility carriers" or "spy nodes". A fertility carrier or spy node F_j^i is the j^{th} chosen node at level i who is allowed to store the secret y_i in its storage. A spy node is used by another same level node to delegate "valid" keys for its descendants. There are at least ℓ such chosen nodes (one at each level) in the hierarchy. A spy node is assumed to be non-malicious and uncompromising like the PKG.

- **Key Delegation in Spy nodes:** This algorithm is executed by the PKG. With the identity of a spy node F_j^L $(1 \leq L \leq \ell)$ as input, this algorithm outputs the key y_L.

- **Key Delegation at Level 1 ($\mathcal{D}_1$):** This algorithm is executed by the PKG. This algorithm works exactly same as the $\mathcal{D}_1$ of α-BSOK.

- **Shared Key Computation at level 1 ($\mathcal{SHK}_1$):** This algorithm works exactly same as Shared Key algorithm $\mathcal{SHK}_1$ of α-BSOK.

- **Key Delegation at level L ($\mathcal{D}_L$, $2 \leq L \leq \ell$):** This algorithm is executed by a node in the level $L-1$ to generate keys for its children in level L. Let us assume m^{th} node in level $L-1$ is the parent of the j^{th} node in level L. With input the secret key set $\mathcal{K}_{(L-1)m}$ and the identities ID_m^{L-1} and ID_j^L, this algorithm runs the $\mathcal{D}_L$ algorithm of α-BSOK which outputs a set $\mathcal{K}''_{Lj}$ containing n_{L-1} elements from $\mathbb{G}$. Here n_i represents total number of nodes from first level to the i^{th} level of the hierarchy $(n_0 = 0, n_1 = n)$. Now, the algorithm chooses an $r \in_R \mathbb{Z}_q^*$ and returns r

with a set -

$$\mathcal{K}'_{Lj} = \{r \cdot k \;\forall\; k \in \mathcal{K}''_{Lj}\}.$$

At this point, ID_m^{L-1} sends the set $\mathcal{K}'_{Lj}$ to a spy node (say, F_u^{L-1}) of its level. The spy node F_u^{L-1} responds back a set $\tilde{\mathcal{K}}_{Lj} = \{y_{L-1} \cdot k \;\forall\; k \in \mathcal{K}'_{Lj}\}$ to ID_m^{L-1}.
The final secret key set of ID_j^L is now defined as -

$$\mathcal{K}_{Lj} = \{r^{-1} \cdot k \;\forall\; k \in \tilde{\mathcal{K}}_{Lj}\}.$$

This secret key set contains n_{L-1} elements of group $\mathbb{G}$.

- **Remaining Key Generation at level L ($\mathcal{D}_L^*$, $1 \leq L < \ell$):** This algorithm is executed by a node in the level L. Let us assume that the set of remaining keys for a node ID_j^L is denoted by $\mathcal{K}'_{Lj}$ then we define $\mathcal{K}'_{Lj}$ as -

$$\mathcal{K}'_{Lj} = \bigcup_{i=L}^{\ell-1} \bigcup_{k=n_{i-1}}^{n_i} \{H_2[\mathcal{SHK}'(ID_j^L, ID_k^i)] \cdot y_i Q_{ID_j^L}\}$$

where $\mathcal{SHK}'$ is the shared key algorithm of α-BSOK. Now the total number of keys in ID_j^L's key space is -

$$|\mathcal{K}_{Lj}| + |\mathcal{K}'_{Lj}| = n_{L-1} + (n_{\ell-1} - n_{L-1}) = n_{\ell-1}$$

Note that any parent node at level L is required to communicate with lower levelled spy nodes to successfully generate its remaining key set. However, this is a one time cost and $(\ell - L)$ communications suffice this process. After a successful key generation, a parent can do the shared key computation with any other node in the network.

- **Shared Key Computation at level L ($\mathcal{SHK}_L$, $2 \leq L \leq \ell$):** This algorithm works exactly same as Shared Key algorithm $\mathcal{SHK}_L$ of α-BSOK.

- **Shared Key Computation between levels L_1 and L_2 ($\mathcal{SHK}_{L_1L_2}$):** This algorithm is used to compute shared secret key between a pair consisting one node from level L_1 and another from level L_2 where $1 \leq L_1 < L_2 \leq \ell$. Suppose two nodes, say $ID_a^{L_2}$ and $ID_b^{L_1}$ want to compute a shared secret key.

  At $ID_a^{L_2}$'s end:

  $ID_a^{L_2}$ checks its secret key set $\mathcal{K}_{L_2a}$ for the group element it received by the $\mathcal{D}_{L_2}$ algorithm as a multiplication of $ID_b^{L_1}$ and its own parent's shared secret key's H_2

hash with the key y_{L_2-1} and H_1 hash of its identity i.e.,

$$H_2[\mathcal{SHK'}_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})] \cdot y_{L_2-1}Q_{ID_a^{L_2}}.$$

Now, the shared secret key is computed as -

$$\mathcal{SHK'}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1}) = e(x_1 \cdot Q_{ID_a^{L_2}}, Q_{ID_b^{L_1}})$$

$$\text{where } x_1 = H_2[\mathcal{SHK'}_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})] \cdot y_{L_2-1}$$

$$\mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1}) = H_3\Big[\mathcal{SHK'}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})\Big]$$

$$\left\{ \text{For } L_1 = L_2 - 1, \text{ we define } \mathcal{SHK'}_{L_1(L_2-1)}(X,Y) = \mathcal{SHK'}_{L_1}(X,Y) \right\}$$

At $ID_b^{L_1}$'s end:
Since $ID_b^{L_1}$ have $H_2[\mathcal{SHK'}_{L_1(L_2-1)}(ID_b^{L_1}, parent(ID_a^{L_2}))] \cdot y_{L_2-1}Q_{ID_b^{L_1}}$ stored in its storage, therefore its shared secret key with $ID_a^{L_2}$ can be computed as -

$$\mathcal{SHK'}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) = e(x_2 \cdot Q_{ID_b^{L_1}}, Q_{ID_a^{L_2}})$$

$$\text{where } x_2 = H_2[\mathcal{SHK'}_{L_1(L_2-1)}(ID_b^{L_1}, parent(ID_a^{L_2}))] \cdot y_{L_2-1}$$

$$\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) = H_3\Big[\mathcal{SHK'}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})\Big]$$

Correctness: In the above computation, we can notice that at ID_a's end, the stored corresponding secret key was computed by its parent in a similar manner to ID_b's computation of $x_2 \cdot Q_{ID_b^{L_1}}$ in D_L^*, therefore, from the correctness argument of the $\mathcal{SHK}_{L_1}$, we have -

$$x_1 = x_2$$
$$\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2}) = \mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})$$

Note that $\mathcal{SHK'}_{L_1L_2}$ is used as shared secret in key delegation for any lower level descendant.

Remark: Similar to α-BSOK, here also we can replace h, H_2, H_3 hash functions by a single hash function H.

4.2.2 Security of β-BSOK

We argue the security for β-BSOK protocol in the H-IND-SK model.

Security Proof

Since β-BSOK is a hybrid of two different protocols, we show the proof of security using the following two cases:

- **Case 1:** When both the target nodes are at level 1: We provide the proof for this case using the security of BIOS, which is the underlying protocol at level 1.
- **Case 2:** When at least one of the target nodes lies beyond level 1: We provide the proof for this case using the S-DBDH assumption.

It is easy to notice that these two cases contain all possible scenarios. As according to these two cases, we consider two different types of adversaries against β-BSOK protocol.

- **Case 1:** (Both the target nodes are at level 1) The proof for this case follows in a similar way to the proof of Case 1 in the Security of α-BSOK (see Section 4.1.2).
- **Case 2:** (When at least one of the target nodes lies beyond level 1; let the lower levelled target node lies at level m in the hierarchy; $1 < m \leq \ell$) We define the restrictions over the adversary so that it does not win the security game trivially in Case 2 as follows:
    1. Adversary is not allowed to compromise the target nodes.
    2. Adversary is not allowed to compromise spy nodes from levels which are higher than any of the target node's level.
    3. Adversary is not allowed to ask for the shared secret key between the target nodes.

We show how to construct an efficient S-DBDH adversary $\mathcal{B}$, using an H-IND-SK adversary $\mathcal{A}_2$ which has a non-negligible advantage against β-BSOK protocol in Case 2. The game between the S-DBDH challenger $\mathcal{C}$ and its adversary $\mathcal{B}$ starts with $\mathcal{C}$ first providing $\mathcal{B}$ the S-DBDH instance (aP, cP, Q) with $(\mathbb{G}, \mathbb{G}_T, e, q, P)$. Note that $\mathcal{B}$ does not have any control over $\mathcal{C}$. $\mathcal{B}$ only seeks the help of $\mathcal{C}$ to generate the instance of the hard problem. Now, $\mathcal{B}$ who is the challenger for β-BSOK protocol interacts with $\mathcal{A}_2$ in the following manner:

- **Setup:** $\mathcal{B}$ gives $mpk = (\mathbb{G}, \mathbb{G}_T, e, P, h, H_1, H_2, H_3)$ to $\mathcal{A}_2$, where h, H_1, H_2 and H_3 are random oracles controlled by $\mathcal{B}$. $\mathcal{A}_2$ can make the following queries:

- **h-queries:** $\mathcal{B}$ receives queries of the form $x_{ij} = (K_i || ID_{1j})$ from $\mathcal{A}_2$. In response to this query, $\mathcal{B}$ maintains an h-table with entries of the form $(x_{ij}, h(x_{ij}))$. If the queried argument is already there in h-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in h-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **H$_1$-queries:** $\mathcal{A}_2$ provides an identity ID_i from the identity space. To Respond such queries $\mathcal{B}$ maintains an H_1-table with entries $(ID_i, s_i, t_i, H_1(ID_i))$ where $t_i \in \{0, 1\}$. If the provided ID_i is already there in the table, $\mathcal{B}$ responds with corresponding $H_1(ID_i)$ entry. Otherwise for some fixed μ, it chooses t_i from $\{0, 1\}$ with probability distribution defined as -

$$\begin{aligned} Pr(t_i = 0) = &\ (1 - \mu) \quad \forall\ ID_i \in \mathcal{ID} \\ Pr(t_i = 1) = &\ \mu \qquad\quad\ \ \forall\ ID_i \in \mathcal{ID}. \end{aligned}$$

  Now, the query by $\mathcal{A}_2$ is responded in the following manner:

  1. If $t_i = 0$ then $\mathcal{B}$ chooses an integer $s_i \in_R \mathbb{Z}_q^*$, adds entries $(ID_i, s_i, 0, s_iP)$ to the table and returns $H_1(ID_i) = s_iP$.
  2. If $t_i = 1$ then $\mathcal{B}$ chooses an integer $s_i \in_R \mathbb{Z}_q^*$, adds entries $(ID_i, s_i, 1, s_iaP)$ to the table and returns $H_1(ID_i) = s_iaP$.

- **H$_2$-queries:** $\mathcal{B}$ receives an element γ from $\mathbb{G}_T$ as query. In response to this query, $\mathcal{B}$ maintains an H_2-table with entries of the form $(\gamma, H_2(\gamma))$. If the queried argument is already there in H_2-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Else, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_2-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **H$_3$-queries:** $\mathcal{B}$ receives an element κ either from $\mathbb{G}_T$ or from $\mathbb{Z}_q^*$ as query. In response to this query, $\mathcal{B}$ maintains an H_3-table with entries of the form $(\kappa, H_3(\kappa))$. If the queried argument is already there in H_3-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_3-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **Extract queries:** $\mathcal{A}_2$ provides an identity ID_i. $\mathcal{B}$ makes an H_1 hash query on ID_i, if this has not been queried already. $\mathcal{B}$ maintains an SK-table with entries of the form $((ID_u, ID_v), r_{uv})$ and responds to the extract query in the following manner:

    1. If $t_i = 1$, then $\mathcal{B}$ aborts the simulation.
    2. If $t_i = 0$ and $ID_i \notin \mathcal{ID}^1$, then $\mathcal{B}$ first finds an entry $(ID_i, s_i, 0, s_iP)$ in the H_1-table. It then checks SK-table for r_{uv} values corresponding to the pairs whose shared keys are being used to compute secret keys of ID_i in β-BSOK. If such an entry is not there in the SK-table, $\mathcal{B}$ chooses an element uniformly at random from $\mathbb{Z}_q^*$ and saves the entry in the table. Now, with these r_{uv} values, $\mathcal{B}$ computes the set $\{s_i(r_{uv}cP)\}$ and sends it to $\mathcal{A}_2$ as the secret key set of ID_i. Note that the number of keys in the output set is $n_{\ell-1}$.
    3. If $t_i = 0$ and $ID_i \in \mathcal{ID}^1$, then $\mathcal{B}$ first finds an entry $(ID_i, s_i, 0, s_iP)$ in the H_1-table. It then checks SK-table for r_{uv} values corresponding to the pairs whose shared keys are being used to compute secret keys of ID_i in β-BSOK. If such an entry is not there in the SK-table, $\mathcal{B}$ chooses an element uniformly at random from $\mathbb{Z}_q^*$ and saves the entry in the table. Now, with these r_{uv} values, $\mathcal{B}$ computes the set $\{s_i(r_{uv}cP)\}$. Note that the number of keys in this set is $n_{\ell-1}$. $\mathcal{B}$ also generate another set consisting $\frac{n_1+1}{2}$ elements from $\mathbb{Z}_q^*$ either chosen uniformly at random or defined using previous queries. It sends these two sets to $\mathcal{A}_2$ as the secret key set of ID_i.

  All extract queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Reveal queries:** $\mathcal{A}_2$ provides an identity pair $\{ID_i, ID_j\}$. $\mathcal{B}$ makes H_1 hash queries on the identities ID_i and ID_j if this has not been already done. $\mathcal{B}$ responds to the reveal query in the following manner:

    1. If $t_i = t_j = 1$, then $\mathcal{B}$ aborts the simulation.
    2. If $t_i \neq t_j$ then $\mathcal{B}$ picks one identity (say, ID_i) with $t_i = 0$, extracts its secret key set and finds the entry $s_i(r_{uv}cP)$ such that $\{parent(ID_i), parent(ID_j)\} = \{ID_u, ID_v\}$. It then responds $H_3\Big(e\big(s_i(r_{uv}cP), H_1(ID_j)\big)\Big)$ to $\mathcal{A}_2$ as the shared secret key.

  All reveal queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Test query:** $\mathcal{A}_2$ makes a single Test query on a pair of identities (say, ID_x and ID_y). If $\{t_x, t_y\} \neq \{1, 1\}$ then $\mathcal{B}$ aborts the simulation and fails. Otherwise $\mathcal{B}$

first finds the entries $(ID_x, s_x, 1, s_xaP)$ and $(ID_y, s_y, 1, s_yaP)$ in the H_1-table and then for the parent pair of the target identities, say (ID_f, ID_g), it finds the entry $((ID_f, ID_g), r_{fg})$ in the SK-table and outputs $H_3(Q^{s_xs_yr_{fg}}) \in \mathbb{Z}_q^*$ as the shared secret key for the challenge identities.

In the end, $\mathcal{A}_2$ outputs a guess bit $\delta' \in \{0, 1\}$.

Note that if $\delta = 1$ i.e., the S-DBDH instance really contained $e(P,P)^{a^2c}$ as Q, then

$$H_3(Q^{s_xs_yr_{fg}}) = H_3\Big(e(P,P)^{a^2cs_xs_yr_{fg}}\Big) = H_3\Big(e\big(s_xaP, r_{fg}c(s_yaP)\big)\Big) = SHK(ID_x,\ ID_y).$$

i.e., here Q corresponds to the real shared secret key between the challenge identities.

On the other hand, if $Q = e(P,P)^r$, for some random $r \in \mathbb{Z}_q^*$, then the test query response by $\mathcal{B}$ is a random element from $\mathbb{Z}_q^*$ which corresponds to $\delta' = \delta = 0$.

In other words, $\mathcal{B}$'s response is correct whenever $\mathcal{A}_2$'s response is correct, which means the advantage of $\mathcal{B}$ against S-DBDH problem is same as the advantage of $\mathcal{A}_2$ against the β-BSOK protocol, except in the events of abort of $\mathcal{B}$.

Analysis: Let $\mathcal{U}$ be the event that $\mathcal{B}$ does not abort during its simulation under H-IND-SK model. Let us denote the number of **Extract** and **Reveal** queries by q_E and q_R respectively. To formally analyze the advantage of $\mathcal{A}_2$ against β-BSOK, we define the following events:

$U_1 = \bigwedge_{i=1}^{q_E}(t_i = 0)$,

$U_2 = \bigwedge_{j=1}^{q_R}\big((t_{j1} = 0) \vee (t_{j2} = 0)\big)$ where (ID_{j1}, ID_{j2}) is the j^{th} reveal query,

$U_3 = \big((t_x = 1) \wedge (t_y = 1)\big)$.

One can easily see that $Pr(\mathcal{U}) = Pr(U_1 \cap U_2 \cap U_3)$. Thus, we have -

$$\begin{aligned} & Pr(\mathcal{U}) = Pr(U_1) \cdot Pr(U_2 \cap U_3 | U_1) \\ \implies & Pr(\mathcal{U}) \geq \mu^{q_E} \cdot \mu^{q_R}(1-\mu)^2. \end{aligned}$$

For $f(\mu) = \mu^{q_E} \cdot \mu^{q_R}(1-\mu)^2$, it is easy to show that f achieves its maximum value at $\mu_0 = 1 - \frac{2}{2+q_E+q_R}$. Therefore, for $\mu = \mu_0$, we have -

$$Pr(\mathcal{U}) \geq \frac{4}{(2+q_E+q_R)^2}\left(1 - \frac{2}{2+q_E+q_R}\right)^{q_E+q_R}$$

$$\implies Pr(\mathcal{U}) \geq \frac{4}{(q_E+q_R)^2}\left[\left(1 - \frac{2}{2+q_E+q_R}\right)^{-\frac{2+q_E+q_R}{2}}\right]^{-2}.$$

Using the estimation $\lim_{x\to 0}(1+x)^{\frac{1}{x}} = e$ and assuming $q_E \approx q_R$, we can define the maximum value of this lower bound for $Pr(\mathcal{U})$ as -

$$\implies Pr(\mathcal{U}) \geq \frac{4}{e^2(q_E+q_R)^2} \approx \frac{1}{(eq_E)^2}.$$

If the advantage of $\mathcal{A}_2$ against β-BSOK protocol is $Adv_{\beta\text{-}BSOK}$, then the advantage $Adv_{S\text{-}DBDH}$ for $\mathcal{B}$ against S-DBDH problem can be given as:

$$Adv_{S\text{-}DBDH} \geq \frac{Adv_{\beta\text{-}BSOK}}{(eq_E)^2}.$$

Since q_E is polynomial in number, if $Adv_{\beta\text{-}BSOK}$ is non-negligible, then we can see that $Adv_{S\text{-}DBDH}$ is also non-negligible which contradicts the hardness assumption of S-DBDH problem. Hence, $Adv_{\beta\text{-}BSOK}$ must be negligible. □

4.2.3 Tighter Security Reductions

A proof of security for a given protocol usually begins by a reduction showing how an adversary breaking the protocol in polynomial time can be used to solve some underlying hard problem in polynomial time. These reductions can be *asymptotic* i.e., they only guarantee that, as the security parameter increases, no polynomial time adversary can break the protocol with a non-negligible probability. However, such a result provides limited information about the security of a protocol *in practice* with a specific key size and against adversaries with certain amount of computational capabilities. Thus, for practical considerations it is important to focus on *concrete* security reductions which give precise bounds on an adversary's success probability as a function of its expended resources.

The most efficient reduction what we expect is one in which an adversary who breaks a protocol in time t with probability ϵ can be used to construct another adversary who can break the underlying hard problem in time $t' \approx t$ with probability $\epsilon' \approx \epsilon$. Such a reduction is called a *tight* reduction. A protocol with a non-tight security reduction will definitely require larger parameter sizes to provide the same security guarantee compared to the case where a tight reduction is known for the same protocol. In fact, it is possible in some cases when a reasonable security assurance from a protocol with a non-tight reduction requires very large key length which is totally impractical.

Katz and Wang provided a modification technique in [19] for some protocols to get a tighter reduction. Full domain hash [3, 4], BLS signatures [7] and Boneh-Franklin ID-based encryption [6] are some examples of such protocols. We have realized that the Katz-Wang technique is not directly applicable in the security models where key escrows are allowed to be compromised, therefore, we can not use this technique on α-BSOK to get a tighter reduction. However, the same is applicable on β-BSOK. In next section, we present the variant of β-BSOK with a tighter security reduction using Katz-Wang technique.

4.3 β-BSOK with Katz-Wang Technique

In this section, we show how to apply Katz-Wang technique on β-BSOK to achieve tighter security reduction on the S-DBDH problem under H-IND-SK model. We also describe benefits and issues in using Katz-Wang technique.

4.3.1 Protocol description of β-BSOK-KWT

For the description below, we are considering an ℓ levelled hierarchy with n nodes at level 1 and maximum branching factor b_{max} from level 1 to $\ell - 2$. In other words, all nodes in level 1 to $\ell - 2$ can have at most b_{max} children whereas PKG has n children. On the other hand, nodes at level $\ell - 1$ have no branching restrictions i.e. they can have arbitrarily large number of children (leaf nodes).

We provide the construction of β-BSOK-KWT protocol using the protocol description of α-BSOK and β-BSOK. We can define β-BSOK-KWT protocol using the following algorithms:

- **Setup:** This algorithm is executed by the PKG.

With input security parameter 1^λ, number of nodes in level 1 (n) and hierarchy depth (ℓ), this algorithm runs the Setup algorithms of BIOS and SOK protocols. The algorithm outputs master secret key *msk* as two key pools $\{K_0, K_1, \dots, K_{n-1}\}$ and $\{y_1, y_2, \dots, y_\ell\}$ from $\mathbb{Z}_q^*$ and public parameters *mpk* as $(\mathbb{G}, \mathbb{G}_T, e, q, n, h, H_1, H_2, H_3)$, where -

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T,\ h : \{0,1\}^* \to \mathbb{Z}_q^*,$$

$$H_1 : \{0,1\}^* \to \mathbb{G},\ H_2 : \mathbb{G}_T \to \mathbb{Z}_q^* \text{ and } H_3 : \{0,1\}^* \to \mathbb{Z}_q^*.$$

- **Key Delegation in Spy nodes:** This algorithm is executed by the PKG. With the identity of a spy node F_j^L $(1 \le L \le \ell)$ as input, this algorithm outputs the key y_L.
- **Key Delegation at Level 1 ($\mathcal{D}_1$):** This algorithm is executed by the PKG. This algorithm works exactly same as the $\mathcal{D}_1$ of β-BSOK.
- **Shared Key Computation at level 1 ($\mathcal{SHK}_1$):** This algorithm works exactly same as Shared Key algorithm $\mathcal{SHK}_1$ of β-BSOK.
- **Key Delegation at level L ($\mathcal{D}_L$, $2 \le L < \ell$):** This algorithm is executed by a node in the level $L-1$ to generate keys for its children in level L. Let us assume m^{th} node in level $L-1$ is the parent of the j^{th} node in level L and n_i represents total number of nodes from first level to the i^{th} level of the hierarchy $(n_0 = 0, n_1 = n)$. Let us consider $Q_{ID_j^L}$ and $Q_{ID_{j,t}^L}$ as *primary* and *secondary* hashes of ID_j^L respectively. We define $Q_{ID_j^L}$ and $Q_{ID_{j,t}^L}$ as -

$$Q_{ID_j^L} = H_1(ID_j^L)$$
$$Q_{ID_{j,t}^L} = \begin{cases} H_1(0||ID_j^L) \text{ if } t = 0 \\ H_1(1||ID_j^L) \text{ if } t = 1 \end{cases}$$

  At first, with $\mathcal{K}_{(L-1)m}, ID_m^{L-1}$ and ID_j^L as input, this algorithm runs shared key algorithm $\mathcal{SHK}'$ of β-BSOK to generate involved parent node ID_m^{L-1}'s *temp-shared* key set say $\{d_1, \dots, d_{n_{L-1}}\}$ where d_i is defined as -

$$d_i = \begin{cases} \mathcal{SHK}'_{r(L-1)}(ID_m^{L-1}, ID_i^r) \text{ if } i \in (n_{r-1}, n_r] \text{ and } 1 \le r \ne L-1, \\ \mathcal{SHK}'_{L-1}(ID_m^{L-1}, ID_i^{L-1}) \text{ if } i \in (n_{L-2}, n_{L-1}]. \end{cases}$$

All these temp-shared keys belong to $\mathbb{G}_T$ or $\mathbb{Z}_q^*$ (depending upon the type of parent's shared secret keys). At this point, the algorithm maps these shared secret keys to $\mathbb{Z}_q^*$ using the hash function H_2. Once it gets the set of these mapped shared secret keys, it chooses a random element $r \in \mathbb{Z}_q^*$, a random bit $t \in \{0,1\}$ and outputs r and t with two sets $\{rH_2(d_1)Q_{ID_j^L}, \ldots\ldots, rH_2(d_{n_{L-1}})Q_{ID_j^L}\}$ and $\{rH_2(d_1)Q_{ID_{j,t}^L}, \ldots\ldots, rH_2(d_{n_{L-1}})Q_{ID_{j,t}^L}\}$. These sets are then converted into two final (primary and secondary) secret key sets with the help of any spy node at level $L-1$. The primary secret key set of ID_j^L is defined as $\{\, y_{L-1}H_2(d_1)Q_{ID_j^L}, \ldots \ldots, y_{L-1}H_2(d_{n_{L-1}})Q_{ID_j^L}\}$ and the secondary secret key set of ID_j^L is defined as $\{t,\, y_{L-1}H_2(d_1)Q_{ID_{j,t}^L}, \ldots\ldots, y_{L-1}H_2(d_{n_{L-1}})Q_{ID_{j,t}^L}\}$. These secret key sets contain a total of $2n_{L-1}$ elements from group $\mathbb{G}$ and a random bit t. We define these sets as the secret key sets $\mathcal{K}_{Lj}$ and $\mathcal{K}_{L(j,t)}$ respectively. Note that here t is redefined for each delegation.

- **Key Delegation at level ℓ ($\mathcal{D}_\ell$):** This algorithm is executed by a node in the level $\ell-1$ to generate keys for its children in level ℓ. Let us assume m^{th} node in level $\ell-1$ is the parent of the j^{th} node in level ℓ. At first, with $\mathcal{K}_{(\ell-1)m}, ID_m^{\ell-1}$ and ID_j^ℓ as input, this algorithm runs shared key algorithm $\mathcal{SHK}'$ of β-BSOK to generate involved parent node $ID_m^{\ell-1}$'s *temp-shared* key set say $\{d_1, \ldots, d_{n_{\ell-1}}\}$ where d_i is defined as -

$$d_i = \begin{cases} \mathcal{SHK}'_{r(\ell-1)}(ID_m^{\ell-1}, ID_i^r) \text{ if } i \in (n_{r-1}, n_r] \text{ and } 1 \le r \ne \ell-1, \\ \mathcal{SHK}'_{\ell-1}(ID_m^{\ell-1}, ID_i^{\ell-1}) \text{ if } i \in (n_{\ell-2}, n_{\ell-1}]. \end{cases}$$

  All these temp-shared keys belong to $\mathbb{G}_T$ or $\mathbb{Z}_q^*$ (depending upon the type of parent's shared secret keys). At this point, the algorithm maps these shared secret keys to $\mathbb{Z}_q^*$ using the hash function H_2. Once it gets the set of these mapped shared secret keys, it chooses a random element $r \in \mathbb{Z}_q^*$, a random bit $t \in \{0,1\}$ and outputs r and t with a set $\{rH_2(d_1)Q_{ID_{j,t}^\ell}, \ldots\ldots, rH_2(d_{n_{\ell-1}})Q_{ID_{j,t}^\ell}\}$. This set is then converted into a final secret key set with the help of any spy node at level $\ell-1$. The final secret key set of ID_j^ℓ is defined as $\{t,\, y_{\ell-1}H_2(d_1)Q_{ID_{j,t}^\ell}, \ldots\ldots, y_{\ell-1}H_2(d_{n_{\ell-1}})Q_{ID_{j,t}^\ell}\}$. This secret key set contains $n_{\ell-1}$ elements from group $\mathbb{G}$ and a random bit t. We define this set as the secret key set $\mathcal{K}_{\ell(j,t)}$. Note that t is redefined for each delegation.

- **Remaining Key Generation at level L ($\mathcal{D}_L^*$, $1 \le L < \ell$):** This algorithm is executed by a node in the level L. Let us assume that the set of remaining keys for a node ID_j^L is denoted by $\mathcal{K}'_{Lj}$ then we define $\mathcal{K}'_{Lj}$ as -

$$\mathcal{K}'_{Lj} = \bigcup_{i=L}^{\ell-1} \bigcup_{k=n_{i-1}}^{n_i} \{H_2[\mathcal{SHK}'(ID_j^L, ID_k^i)] \cdot y_i Q_{ID^L_{j,t_j}}\}$$

where $\mathcal{SHK}'$ is the shared key algorithm of α-BSOK. Now the total number of keys in ID_j^L's key space is -

$$|\mathcal{K}_{Lj}| + |\mathcal{K}_{L(j,t_j)}| + |\mathcal{K}'_{Lj}| = 2n_{L-1} + (n_{\ell-1} - n_{L-1}) = n_{\ell-1} + n_{L-1}.$$

Note that any parent node at level L is required to communicate with lower levelled spy nodes to successfully generate its remaining key set. However, this is a one time cost and $(\ell - L)$ communications suffice this process.

- **Shared Key Computation at level L ($\mathcal{SHK}_L$, $2 \leq L \leq \ell$):** This algorithm is used to compute shared secret key between a pair consisting nodes of level $L \geq 2$. Suppose two nodes at level L, say ID_a^L and ID_b^L want to compute a shared secret key.

  At ID_a^L's end:

  ID_a^L checks its secondary secret key set $\mathcal{K}_{L(a,t_a)}$ for the group element it received by the $\mathcal{D}_L$ algorithm as a multiplication of their parent's temp-shared key's H_2 hash and y_{L-1} with the H_1 hash of its identity concatenated with its random bit t_a i.e.,

  $$H_2[\mathcal{SHK}'_{L-1}(parent(ID_a^L), parent(ID_b^L))] \cdot y_{L-1} Q_{ID^L_{a,ta}}$$

  where $\mathcal{SHK}'$ is the shared key algorithm of α-BSOK. Now, the possible shared secret keys are computed as -

  $$\mathcal{SHK}_L(ID_a^L, ID_b^L)|_{(t_a,0)} = H_3[e(Q_{ID^L_{a,ta}}, Q_{ID^L_{b,0}})^{x_1}]$$

  $$\mathcal{SHK}_L(ID_a^L, ID_b^L)|_{(t_a,1)} = H_3[e(Q_{ID^L_{a,ta}}, Q_{ID^L_{b,1}})^{x_1}]$$

  $$\text{where } x_1 = H_2[\mathcal{SHK}'_{L-1}(parent(ID_a^L), parent(ID_b^L))] \cdot y_{L-1}$$

  At ID_b^L's end:

  ID_b^L checks its secondary secret key set $\mathcal{K}_{L(b,t_b)}$ for the group element it received by the $\mathcal{D}_L$ algorithm as a multiplication of their parent's temp-shared key's H_2 hash and y_{L-1} with the H_1 hash of its identity concatenated with its random bit

t_b i.e.,

$$H_2[\mathcal{SHK}'_{L-1}(parent(ID_b^L), parent(ID_a^L))] \cdot y_{L-1}Q_{ID^L_{b,t_b}}$$

where $\mathcal{SHK}'$ is the shared key algorithm of α-BSOK. Now, the possible shared secret keys are computed as -

$$\mathcal{SHK}_L(ID_b^L, ID_a^L)|_{(0,t_b)} = H_3[e(Q_{ID^L_{b,t_b}}, Q_{ID^L_{a,0}})^{x_2}]$$

$$\mathcal{SHK}_L(ID_b^L, ID_a^L)|_{(1,t_b)} = H_3[e(Q_{ID^L_{b,t_b}}, Q_{ID^L_{a,1}})^{x_2}]$$

$$\text{where } x_2 = H_2[\mathcal{SHK}'_{L-1}(parent(ID_b^L), parent(ID_a^L))] \cdot y_{L-1}$$

Correctness: From the correctness argument of BIOS, we have, $x_1 = x_2$ and therefore using the bilinearity property of the pairing e, we have -

$$t_a = t_b \iff \mathcal{SHK}_L(ID_a^L, ID_b^L)|_{(t_a,t_a)} = \mathcal{SHK}_L(ID_b^L, ID_a^L)|_{(t_b,t_b)}$$
$$t_a = (1-t_b) \iff \mathcal{SHK}_L(ID_a^L, ID_b^L)|_{(t_a,1-t_a)} = \mathcal{SHK}_L(ID_b^L, ID_a^L)|_{(1-t_b,t_b)}$$

This implies that irrespective to the knowledge of $t_a \overset{?}{=} t_b$, both parties can always compute a pair of keys including a valid shared-secret key. Confirmation of this valid key can be done by sending and verifying digest values of their generated key pairs.

- **Shared Key Computation between levels L_1 and L_2 ($\mathcal{SHK}_{L_1L_2}$):** This algorithm is used to compute shared secret key between a pair consisting one node from level L_1 and another from level L_2 where $1 \le L_1 < L_2 \le \ell$. Suppose two nodes, say $ID_a^{L_2}$ and $ID_b^{L_1}$ want to compute a shared secret key.

  At $ID_a^{L_2}$'s end:

  $ID_a^{L_2}$ checks its secondary secret key set $\mathcal{K}_{L_2a,t_a}$ for the group element it received by the $\mathcal{D}_{L_2}$ algorithm as a multiplication of $ID_b^{L_1}$ and its own parent's temp-shared key's H_2 hash and y_{L_2-1} with H_1 hash of its identity concatenated with its random bit t_a i.e.,

$$H_2[\mathcal{SHK}'_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})] \cdot y_{L_2-1}Q_{ID^{L_2}_{a,t_a}}$$

where $\mathcal{SHK}'$ is the shared key algorithm of α-BSOK. Now, the possible shared secret keys are computed as -

$$\mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})|_{(t_a,0)} = H_3[e(Q_{ID_{a,t_a}^{L_2}}, Q_{ID_{b,0}^{L_1}})^{x_1}]$$

$$\mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})|_{(t_a,1)} = H_3[e(Q_{ID_{a,t_a}^{L_2}}, Q_{ID_{b,1}^{L_1}})^{x_1}]$$

$$\text{where } x_1 = H_2[\mathcal{SHK}'_{L_1(L_2-1)}(parent(ID_a^{L_2}), ID_b^{L_1})] \cdot y_{L_2-1}$$

$$\left\{ \text{For } L_1 = L_2 - 1, \text{ we define } \mathcal{SHK}'_{L_1(L_2-1)}(X,Y) = \mathcal{SHK}'_{L_1}(X,Y) \right\}$$

At $ID_b^{L_1}$'s end:

Since $ID_b^{L_1}$ have $H_2[\mathcal{SHK}'_{L_1(L_2-1)}(ID_b^{L_1}, parent(ID_a^{L_2}))]y_{L_2-1} \cdot Q_{ID_{b,t_b}^{L_1}}$ stored in its storage, therefore, its possible shared secret keys with $ID_a^{L_2}$ can be computed as -

$$\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})|_{(0,t_b)} = H_3[e(Q_{ID_{b,t_b}^{L_1}}, Q_{ID_{a,0}^{L_2}})^{x_2}]$$

$$\mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})|_{(1,t_b)} = H_3[e(Q_{ID_{b,t_b}^{L_1}}, Q_{ID_{a,1}^{L_2}})^{x_2}]$$

$$\text{where } x_2 = H_2[\mathcal{SHK}'_{L_1(L_2-1)}(ID_b^{L_1}, parent(ID_a^{L_2}))] \cdot y_{L-1}$$

Correctness: In the above computation, we can notice that at ID_a's end, the stored corresponding secret key was computed by its parent in a similar manner to ID_b's computation of $x_2 \cdot Q_{ID_{b,t_b}^{L_1}}$ in $D_{L_1}^*$, therefore, from the correctness argument of the $\mathcal{SHK}_{L_1}$, we have -

$$\begin{aligned} t_a = t_b &\iff \mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})|_{(t_a,t_a)} = \mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})|_{(t_b,t_b)} \\ t_a = (1-t_b) &\iff \mathcal{SHK}_{L_1L_2}(ID_a^{L_2}, ID_b^{L_1})|_{(t_a,1-t_a)} = \mathcal{SHK}_{L_1L_2}(ID_b^{L_1}, ID_a^{L_2})|_{(1-t_b,t_b)} \end{aligned}$$

This implies that irrespective to the knowledge of $t_a \overset{?}{=} t_b$, both parties can always compute a pair of keys including a valid shared-secret key. Confirmation of this valid key can be done by sending and verifying digest values of their generated key pairs.

Remark: Similar to α-BSOK, here also we can replace h, H_2, H_3 hash functions by a single hash function H.

4.3.2 Security of β-BSOK-KWT

We argue the security for β-BSOK-KWT protocol in the H-IND-SK model.

Security Proof

Since β-BSOK-KWT is a hybrid of two different protocols, we show the proof of security using the following two cases:

- **Case 1:** When both the target nodes are at level 1: We provide the proof for this case using the security of BIOS, which is the underlying protocol at level 1.
- **Case 2:** When at least one of the target nodes lies beyond level 1: We provide the proof for this case using the S-DBDH assumption.

It is easy to notice that these two cases contain all possible scenarios. As according to these two cases, we consider two different types of adversaries against β-BSOK-KWT protocol.

- **Case 1:** (Both the target nodes are at level 1) The proof for this case follows in a similar way to the proof of Case 1 in the Security of α-BSOK (see Section 4.1.2).
- **Case 2:** (When at least one of the target nodes lies beyond level 1; let the lower levelled target node lies at level m in the hierarchy; $1 < m \leq \ell$) We define the restrictions over the adversary so that it does not win the security game trivially in Case 2 as follows:
  1. Adversary is not allowed to compromise the target nodes.
  2. Adversary is not allowed to compromise spy nodes from levels which are higher than any of the target node's level.
  3. Adversary is not allowed to ask for the shared secret key between the target nodes.

We show how to construct an efficient S-DBDH adversary $\mathcal{B}$, using an H-IND-SK adversary $\mathcal{A}_2$ which has a non-negligible advantage against β-BSOK-KWT protocol in Case 2. The game between the S-DBDH challenger $\mathcal{C}$ and its adversary $\mathcal{B}$ starts with $\mathcal{C}$ first providing $\mathcal{B}$ the S-DBDH instance (aP, cP, Q) with $(\mathbb{G}, \mathbb{G}_T, e, q, P)$. Note that $\mathcal{B}$ does not have any control over $\mathcal{C}$. $\mathcal{B}$ only seeks the help of $\mathcal{C}$ to generate the instance of the hard problem. Now, $\mathcal{B}$ who is the challenger for β-BSOK-KWT protocol interacts with $\mathcal{A}_2$ in the following manner:

- **Setup:** $\mathcal{B}$ gives $mpk = (\mathbb{G}, \mathbb{G}_T, e, P, h, H_1, H_2, H_3)$ to $\mathcal{A}_2$, where h, H_1, H_2 and H_3 are random oracles controlled by $\mathcal{B}$. $\mathcal{A}_2$ can make the following queries:

- **h-queries:** $\mathcal{B}$ receives queries of the form $x_{ij} = (K_i || ID_{1j})$ from $\mathcal{A}_2$. In response to this query, $\mathcal{B}$ maintains an h-table with entries of the form $(x_{ij}, h(x_{ij}))$. If the queried argument is already there in h-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in h-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **H$_1$-queries:** $\mathcal{A}_2$ provides an identity ID_i from the identity space. To Respond such queries $\mathcal{B}$ maintains an H_1-table with entries $(ID_i, t_i, H_1(0||ID_i), H_1(1||ID_i))$ where $t_i \in \{0, 1\}$. If the provided ID_i is already there in the H_1-table, $\mathcal{B}$ responds with corresponding $H_1(0||ID_i)$ and $H_1(1||ID_i)$ entries. Otherwise it chooses two random values $s_i \in_R \mathbb{Z}_q^*$ and $t_i \in_R \{0, 1\}$ and based on the t_i output it store the hash entries in the H_1-table as shown below -

  If $t_i = 0$ then the entries are $(ID_i, 0, s_iP, s_iP + aP)$.
  If $t_i = 1$ then the entries are $(ID_i, 1, s_iP + aP, s_iP)$.

  Now, the stored $H_1(0||ID_i)$ and $H_1(1||ID_i))$ entries are provided to $\mathcal{A}$.

- **H$_2$-queries:** $\mathcal{B}$ receives an element γ from $\mathbb{G}_T$ as query. In response to this query, $\mathcal{B}$ maintains an H_2-table with entries of the form $(\gamma, H_2(\gamma))$. If the queried argument is already there in H_2-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Else, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_2-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **H$_3$-queries:** $\mathcal{B}$ receives an element κ either from $\mathbb{G}_T$ or from $\mathbb{Z}_q^*$ as query. In response to this query, $\mathcal{B}$ maintains an H_3-table with entries of the form $(\kappa, H_3(\kappa))$. If the queried argument is already there in H_3-table, the corresponding hash value is given to $\mathcal{A}_2$ as response. Otherwise, an element is chosen uniformly at random from $\mathbb{Z}_q^*$, the entry is saved in H_3-table and the chosen element is provided to $\mathcal{A}_2$ as response.

- **Extract queries:** $\mathcal{A}_2$ provides an identity ID_i. $\mathcal{B}$ makes an H_1 hash query on ID_i, if this has not been queried already. $\mathcal{B}$ maintains an SK-table with entries of the form $((ID_u, ID_v), r_{uv})$ and responds to the extract query in the following manner:

  1. If $ID_i \notin \mathcal{ID}^1$, then $\mathcal{B}$ first finds the corresponding values of t_i and s_i from the entry of ID_i in the H_1-table. It then checks SK-table for r_{uv} values

corresponding to the pairs whose shared keys are being used to compute secret keys of ID_i in β-BSOK-KWT. If such an entry is not there in the SK-table, $\mathcal{B}$ chooses an element uniformly at random from $\mathbb{Z}_q^*$ and saves the entry in the table. Now, with these r_{uv} values, $\mathcal{B}$ computes the set $\{s_i(r_{uv}cP)\}$ and sends it to $\mathcal{A}_2$ as the secret key set of ID_i. Note that the number of keys in the output set is $n_{\ell-1}$. $\mathcal{B}$ also provides t_i to $\mathcal{A}$.

2. If $ID_i \in \mathcal{ID}^1$, then $\mathcal{B}$ first finds the corresponding values of t_i and s_i from the entry of ID_i in the H_1-table. It then checks SK-table for r_{uv} values corresponding to the pairs whose shared keys are being used to compute secret keys of ID_i in β-BSOK-KWT. If such an entry is not there in the SK-table, $\mathcal{B}$ chooses an element uniformly at random from $\mathbb{Z}_q^*$ and saves the entry in the table. Now, with these r_{uv} values, $\mathcal{B}$ computes the set $\{s_i(r_{uv}cP)\}$. Note that the number of keys in this set is $n_{\ell-1}$. $\mathcal{B}$ also generate another set consisting $\frac{n_1+1}{2}$ elements from $\mathbb{Z}_q^*$ either chosen uniformly at random or defined using previous queries. It sends these two sets along with t_i to $\mathcal{A}_2$ as the secret key set of ID_i.

All extract queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Reveal queries:** $\mathcal{A}_2$ provides an identity pair $\{ID_i, ID_j\}$. $\mathcal{B}$ makes H_1 hash queries on the identities ID_i and ID_j if this has not been already done. $\mathcal{B}$ picks any one of the two queried identity (say, ID_j), extracts its secret key set and finds the entry $s_j(r_{uv}cP)$ such that $\{ID_u, ID_v\} = \{parent(ID_i), parent(ID_j)\}$. $\mathcal{B}$ then responds to the reveal query with $SHK(ID_i, ID_j) = H_3\Big(e\big(s_iP, s_j(r_{uv}cP)\big)\Big)$.

  Note that $SHK(ID_i, ID_j) = H_3\Big(e\big(s_iP, s_j(r_{uv}cP)\big)\Big)$ is always one of the correct responses disregarding of whether any previous extractions are done on ID_i and ID_j or not. However, if a **Reveal query** on an identity pair is made before the **Extract query** on either of them, then $\mathcal{B}$ first makes **Extract queries** on them and updates all the relevant tables to maintain consistency and correctness among its responses.

  All reveal queries are defined while maintaining consistency and correctness with previous extract and reveal queries.

- **Test query:** $\mathcal{A}_2$ makes a single Test query on a pair of identities (say, ID_x and ID_y). $\mathcal{B}$ first finds the values of s_x and s_y from the corresponding entries of ID_x and ID_y in the H_1-table and then for the parent pair of the target identities, say

(ID_f, ID_g), it finds the entry $((ID_f, ID_g), r_{fg})$ in the SK-table. $\mathcal{B}$ uses $s_xP + aP$ as the H_1 hash for ID_x and $s_yP + aP$ as H_1 hash for ID_y and responds with $SHK(ID_x, ID_y) = H_3\big(Q^{r_{fg}} \cdot e(P, cP)^{r_{fg}s_xs_y} \cdot e(aP, cP)^{r_{fg}(s_x+s_y)}\big)$.

In the end, $\mathcal{A}_2$ outputs a guess bit $\delta' \in \{0, 1\}$.

Note that if $\delta = 1$ i.e., the S-DBDH instance really contained $e(P, P)^{a^2c}$ as Q, then

$$\begin{aligned}
& H_3\big(Q^{r_{fg}} \cdot e(P, cP)^{r_{fg}s_xs_y} \cdot e(aP, cP)^{r_{fg}(s_x+s_y)}\big) \\
&= H_3\big(e(P, P)^{a^2cr_{fg}} \cdot e(P, P)^{cr_{fg}s_xs_y} \cdot e(P, P)^{acr_{fg}(s_x+s_y)}\big) \\
&= H_3\big(e(P, P)^{cr_{fg}(a^2+s_xs_y+a(s_x+s_y))}\big) \\
&= H_3\Big(e\big((s_x + a)P, r_{fg}c(s_y + a)P\big)\Big) \\
&= H_3\Big(e\big((s_xP + aP), r_{fg}c(s_yP + aP)\big)\Big) \\
&= SHK(ID_x,\ ID_y).
\end{aligned}$$

i.e., here Q corresponds to the real shared secret key between the challenge identities.

On the other hand, if $Q = e(P, P)^r$, for some random $r \in \mathbb{Z}_q^*$, then the test query response by $\mathcal{B}$ is a random element from $\mathbb{Z}_q^*$ which corresponds to $\delta' = \delta = 0$.

In other words, $\mathcal{B}$'s response is correct whenever $\mathcal{A}_2$'s response is correct, which means the advantage of $\mathcal{B}$ against S-DBDH problem is same as the advantage of $\mathcal{A}_2$ against the β-BSOK-KWT protocol. Formally, if the advantage of $\mathcal{A}_2$ against β-BSOK-KWT protocol is $Adv_{\beta\text{-}BSOK\text{-}KWT}$, then the advantage $Adv_{S\text{-}DBDH}$ for $\mathcal{B}$ against S-DBDH problem can be given as:

$$Adv_{S\text{-}DBDH} = Adv_{\beta\text{-}BSOK\text{-}KWT}.$$

If $Adv_{\beta\text{-}BSOK\text{-}KWT}$ is non-negligible, then we can see that $Adv_{S\text{-}DBDH}$ is also non-negligible which contradicts the hardness assumption of S-DBDH problem. Hence, $Adv_{\beta\text{-}BSOK\text{-}KWT}$ must be negligible. $\square$

4.3.3 Pros and Cons of using Katz-Wang Technique

We can notice that degradation in the security reduction of β-BSOK differs from β-BSOK-KWT by a factor of $e^2q_E^2$. The number of extract queries (q_E) is normally considered around 2^{30} and therefore the degradation is $\approx 2^{63}$. This means if there exists an adversary against S-DBDH problem with an advantage of 2^{-64} then it can be used to create an adversary with an advantage $\approx 1/2$ against β-BSOK protocol. This implies β-BSOK is not secure under such group settings (the security parameter) and to improve this we need to increase the size of underlying public parameters. In other words, to achieve a certain level of security, we need to compromise the efficiency of the protocol. On the other hand, in β-BSOK-KWT, we get the tightest reduction, i.e. for the same adversary against S-DBDH problem with an advantage of 2^{-64}, we can construct an adversary with at most advantage of 2^{-64} (i.e. 64-bit security assurance) for β-BSOK-KWT.

It is clear that Katz-Wang technique helps to improve the efficiency of β-BSOK. However, this benefit comes with the price of double key-storage requirement in non-leaf nodes. Here we also need to do one extra computation and one extra communication to compute and confirm the final shared-secret key between any two involved parties. Still this is a one-time cost and after this confirmation of the shared-key (which can be done by sending and verifying the hash of computed pairs of shared-keys) both parties can directly use the final shared-key for their future communications.

4.4 Summary

In this chapter, we have proposed α-BSOK; a key exchange protocol which is *fully resilient*, *identity-based*, *non-interactive* and *decentralized*. We have also proved that α-BSOK is secure under H-IND-SK model. We have defined β-BSOK as a modified version of α-BSOK to avoid hierarchical key-escrow. We have also showed that Katz-Wang technique can be used over β-BSOK to give a *tighter* reduction. Thus, we can argue that α-BSOK (including its variants) can be a perfect solution for H-ID-NIKE in MANETs.

In upcoming chapters, we will analyze implementation and simulation data of α-BSOK and will do a comparative analysis with HHKAS and other related protocols for efficiency comparison.

Chapter 5

Comparative Analysis

In this chapter, we present a comparative analysis of α-BSOK with HHKAS and other related protocols. From this analysis, we can conclude that α-BSOK performs better in both time and storage requirements among existing H-ID-NIKE protocols.

5.1 Prototype Implementation

Implementation of α-BSOK is done in Python. Akinyele *et al.* introduced a free Python language library known as the Charm-crypto [1] which allows elliptic curve arithmetic and pairing computation for implementation of pairing-based cryptographic protocols. We have used Charm-crypto over 'SS512' pairing (commonly known as Type A pairing which was introduced by Lynn in his Ph.D. thesis [21]) to implement α-BSOK and to analyze its requirements of storage and time. We have also implemented HHKAS over the same security parameters and system configurations to do a fair efficiency comparison between α-BSOK and HHKAS. 'SS512' pairing is a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ constructed on the elliptic curve $y^2 = x^3 + x$ over the field F_q for some prime $q = 3 \bmod 4$. Here $\mathbb{G}$ is the group of points $E(F_q)$ and $\mathbb{G}_T$ is a subgroup of F_{q^2}. The test machine which we used had the following system configuration:

Parameter	Value	Parameter	Value
Vendor	GenuineIntel	Cache size (L1, L2, L3), in KB	(32, 256, 8192)
Model	58	CPU(s)	8
Model name	Intel(R) Core(TM) i7-3770	Core(s)/socket	4
Total Memory (MB)	15800	Architecture	64-bit
VGA Memory (MB)	256+4	Bogomips	6784.85
Max CPU MHz	3400 MHz	OS	Kubuntu 18.04
Avg CPU MHz	2700 MHz	Kernal	Linux, 4.15.0-36-generic

Table 5.1 Test machine configuration.

We have used 'SHA-256'[22] to compute time for all hashings into $\mathbb{Z}_q^*$ whereas 'SS512'[21] pairing is used to compute time for addition in $\mathbb{G}$, hashing into $\mathbb{G}$ and pairing computation into $\mathbb{G}_T$. The storage and time requirements for these computations are shown in the Table 5.2.

Characteristics of Elliptic Curves (Used in implementation)	Experimental Results (Python-3.6.x ,Charm-0.50)
Bitlength of an element in $\mathbb{G}$	128 B
Bitlength of an element in $\mathbb{G}_T$	128 B
Bitlength of an element in $\mathbb{Z}_q$	20 B
Time taken by an addition in $\mathbb{G}$	0.008 ms
Time taken by a hashing into $\mathbb{G}$ (H_1)	2.74 ms
Time taken by a pairing to $\mathbb{G}_T$	0.97 ms
Time taken by a hashing into $\mathbb{Z}_q$ (h)	0.05 ms
Time taken by a hashing into $\mathbb{Z}_q$ (H_2, H_3)	0.14 ms
Time taken by a group element serialization	0.05 ms

Table 5.2 Requirements for operations in 'SS512' pairing.

In HHKAS, i^{th} level threshold t_i is defined as $2^{x_i} - 1$ for some $x_i \in \mathbb{N}^+$ and to make it equally resilient to α-BSOK, we need to put $t_i = N_i - 2$ (i.e. full resilience) where N_i represents number of nodes in i^{th} level. This way we can obtain a special collection $\mathcal{H}$ of hierarchies to do a fair comparison between HHKAS and α-BSOK. We can notice that in all these hierarchies, $N_i = 2^{x_i} + 1$ for some $x_i \in \mathbb{N}^+$.

For all type of analysis below, we have used binary hierarchies up to 5 levels from the collection $\mathcal{H}$ with $x_i = (2 + i)$ i.e the number of nodes in corresponding levels (if exist) are defined as $N_1 = 9, N_2 = 17, N_3 = 33, N_4 = 65$ and $N_5 = 129$. A complete image of this tree structure is shown in the Appendix A.

5.2 HHKAS vs α-BSOK

Efficiency comparison between HHKAS and α-BSOK is presented in the following subsections. The same tree structure is used in HHKAS and α-BSOK's implementation and all factors are analyzed over the same machine with same security parameters therefore the comparison of these data is invariant of platform.

5.2.1 HHKAS vs α-BSOK: Key-Storage

Data in Table 5.3 shows that in comparison to HHKAS, α-BSOK requires very less amount of storage to store the secret keys for nodes in the hierarchy. Here we can infer that in HHKAS, key-storage of a node at any fixed level is exponentially dependant on the length of the hierarchy and drastically increases with the number of levels whereas for α-BSOK key-storage at any fixed level increases by a constant factor $C = b_{\ell-1} - 1$ (here $b_{\ell-1}$ is the branching factor at level $\ell - 1$) with an increase in the number of levels. In this hierarchy $C = 1$. Also, for a fixed hierarchy, α-BSOK key-storage remains same from first level nodes to leaves. This property can be useful in a network containing identical devices for all level nodes.

Hierarchical depth $\rightarrow$	2		3		4		5	
Node at level $\downarrow$	α-BSOK	HHKAS	α-BSOK	HHKAS	α-BSOK	HHKAS	α-BSOK	HHKAS
0 (Root)	252 B	64.9 KB	252 B	42.85 MB	252 B	123.2 GB	252 B	Impractical
1	0.98 KB	10.5 KB	2.6 KB	7.5 MB	5.6 KB	19.8 GB	11.7 KB	Impractical
2	0.84 KB	41.7 KB	2.4 KB	485 KB	5.5 KB	1.26 GB	11.5 KB	Impractical
3	$\times$[1]	$\times$	2.4 KB	1.74 MB	5.5 KB	36.78 MB	11.5 KB	Impractical
4	$\times$	$\times$	$\times$	$\times$	5.5 KB	145.09 MB	11.5 KB	Impractical
5	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	11.5 KB	Impractical

Table 5.3 HHKAS vs α-BSOK: Node Key-Storage requirements for different levelled hierarchies.

5.2.2 HHKAS vs α-BSOK: Key-Generation Time

Similar to key-storage, data in Table 5.4 shows that in comparison to HHKAS, at any parent level, a node in α-BSOK requires very less time to generate keys for its child nodes. We can also infer that in HHKAS, key-generation time for a node at any fixed level is exponentially dependant on the length of the hierarchy and drastically increases with the number of levels whereas for α-BSOK key-generation time at any fixed level increases by the same constant factor $C(= 1)$ defined in the last subsection. Also, for a fixed hierarchy, α-BSOK key-generation time is almost same from first level nodes to leaves whereas in HHKAS key-generation time requirements for first level nodes and for leaves is very high. In fact, HHKAS becomes impractical for moderate hierarchies containing 5 or more levels.

[1]'$\times$' means such node and level doesn't exist in the corresponding hierarchy.
In Table 5.3, red (respectively blue) entry represents the maximum (respectively the minimum) storage requirements for a node in the network.

Hierarchical depth →	2		3		4		5	
Node at level ↓	α-BSOK	HHKAS	α-BSOK	HHKAS	α-BSOK	HHKAS	α-BSOK	HHKAS
0 (Root)	0.0002 Sec	0.0036 Sec	0.0002 Sec	2.7251 Sec	0.0002 Sec	1.69 Hours	0.0002 Sec	Memory Error
1	0.0083 Sec	0.0103 Sec	0.0768 Sec	6.6453 Sec	0.2118 Sec	5.48 Hours	0.4759 Sec	Memory Error
2	0.0067 Sec	1.2360 Sec	0.0821 Sec	0.8385 Sec	0.2173 Sec	40.3 Min	0.4763 Sec	Memory Error
3	×	×	0.1071 Sec	53.3623 Sec	0.2450 Sec	2.42 Min	0.5074 Sec	Memory Error
4	×	×	×	×	0.2478 Sec	1.21 Hours	0.5197 Sec	Memory Error
5	×	×	×	×	×	×	0.5235 Sec	Memory Error

Table 5.4 HHKAS vs α-BSOK: Node Key-Generation Time requirements for different levelled hierarchies.

5.2.3 HHKAS vs α-BSOK: Shared-Key Computation Time

Table 5.5 represents shared-key computation times taken by different type of node pairs for different levelled hierarchies under the two H-ID-NIKE protocols, HHKAS and α-BSOK.
We can clearly see that with α-BSOK, a leaf level node requires comparatively very less time to compute the shared-key with any other leaf node in the network.

Hierarchical depth	Type of shared-key	α-BSOK	HHKAS
3	Leaves Intra-level	0.0040 Sec	7.08 Sec
	Non-leaves Intra-level (max)	0.0040 Sec	–[2]
	Inter-level (max)	0.0041 Sec	–
4	Leaves Intra-level	0.0040 Sec	9.81 Hours
	Non-leaves Intra-level (max)	0.0040 Sec	–
	Inter-level (max)	0.0043 Sec	–
5	Leaves Intra-level	0.0040 Sec	Memory Error
	Non-leaves Intra-level (max)	0.0040 Sec	–
	Inter-level (max)	0.0046 Sec	–

Table 5.5 HHKAS vs α-BSOK: Shared-Key Computation Time for different levelled hierarchies.

Additionally, α-BSOK also supports non-leaves intra-level and all type of inter-level communications. Time requirements for these multi-level shared-key computations in the 5-levelled hierarchy (see Appendix A) are shown in the Table 5.6. Here we can notice that similar to leaf intra-level, all of the non-leaf intra level and inter-level time requirements are very less and feasible.

In Table 5.4, red (respectively blue) entry represents the maximum (respectively the minimum) time requirements to generate secret-keys for a node in the network.

[2]'–' means such shared-key computation in HHKAS is either not possible or not shown to be secure under their presented security model [16].

Node j at level $\rightarrow$ Node i at level $\downarrow$	1 (in Sec)	2 (in Sec)	3 (in Sec)	4 (in Sec)	5 (in Sec)
1	0.0002	0.0041	0.0041	0.0040	0.0041
2	0.0036	0.0036	0.0041	0.0041	0.0041
3	0.0041	0.0041	0.0040	0.0041	0.0041
4	0.0041	0.0043	0.0041	0.0040	0.0041
5	0.0040	0.0043	0.0046	0.0041	0.0040

Table 5.6 Shared-Key Computation Time on different levels in the 5-levelled hierarchy.

5.3 MANET Simulation: Data and Analysis

The primitive parameters for efficiency comparison between two key exchange protocols include key generation time, key storage, shared key computation time and key distribution time. In the Section 5.2, we have done an efficiency analysis of α-BSOK for the first three of four parameters mentioned above. Although the fourth parameter can be eliminated by distributing the keys off-line, there are still many applications such as new node admission or node updation where we need to refresh the keys in the field to ensure the security and connectivity of the network. To analyze likelihood of our protocol in such a MANET scenario, we have done a simulation of α-BSOK using the network simulatior NS-2 [17]. In this section, we present our MANET model and simulation data to analyze key distribution (or refreshment) timings for α-BSOK.

5.3.1 Model

We have defined a general deterministic model for node movement as shown below in Figure 5.1 to mimic the underlying real scenarios. On top of that, we have used the following network parameters which is close to the general MANET configurations.

Parameter	Value	Parameter	Value
Channel type	Wireless	FTP max flow	256 packets
Radio-propagation model	Two-Ray Ground	FTP window size	50 packets
Network interface type	Wireless	Interface max flow	20 packets
MAC type	802_11	Routing protocols	DSR, None
Interface queue type	CMUPriQueue, DropTail	Data packet size (main)	512 bytes
Bandwidth	0.25 M/s	Data packet size (with header)	572 bytes
Frequency	0.4 GHz	Antenna, gain (T, R)	Omni Antenna, (1, 1)
Internet protocol	FTP over TCP	Wireless system loss factor	0.1
TCP window size	20 packets	Mobile nodes	Allowed
Node communication range	250 meter	Simulation topology	(698,500) $\sim$ 0.35 km^2
Simulation time	60 Sec	Network size	16 nodes

Table 5.7 Mobile Ad-hoc Network configuration.

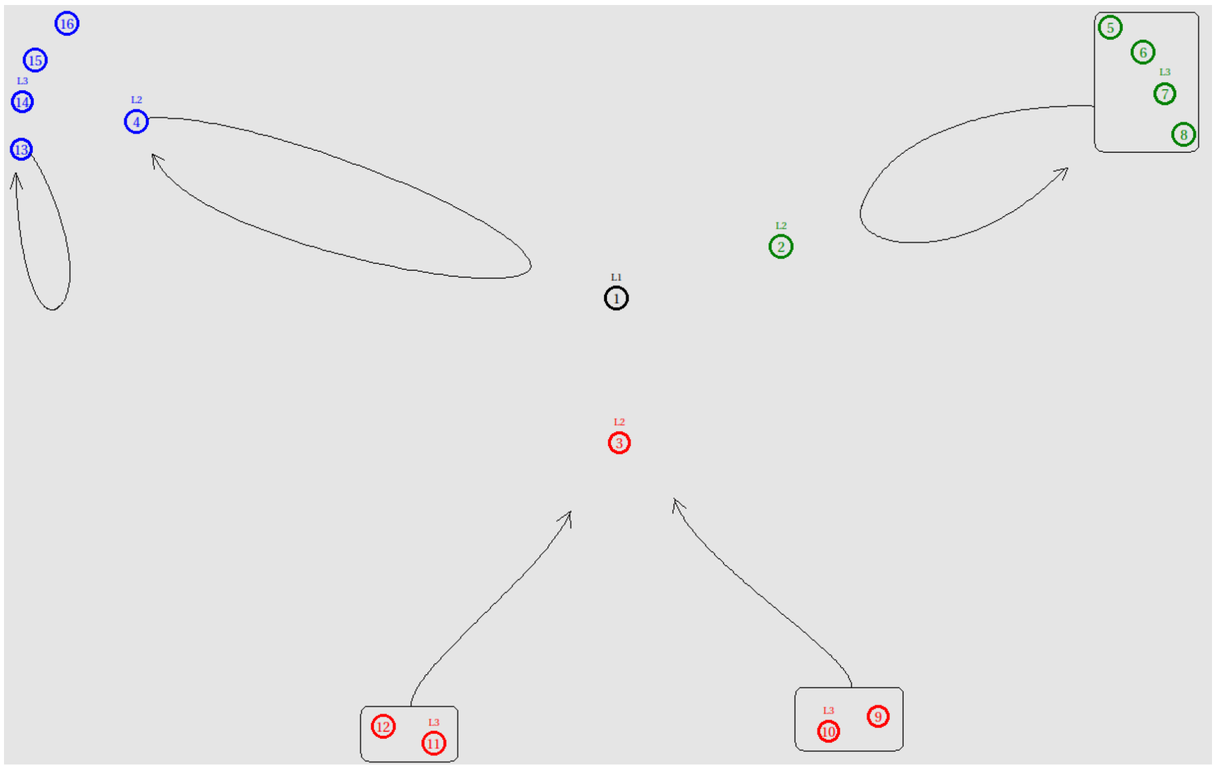

Fig. 5.1 Nodes' mobility model with their hierarchical positions

In this model, we have used a small tactical MANET over three levelled hierarchy. Here a stable central authority is fixed in the center of the MANET deployed region whereas remaining nodes are mobile as shown in the Figure 5.1. Here mobility of the nodes is represented by the curly arrows and rectangular box represents groups with same mobility. Note that in this model, each color represents a 2-levelled subtree of the main hierarchy and mobility is defined in the shown ways just to cover the following possibilities of the real scenarios.

1. After receiving its own keys, a second level node can generate and distribute the new child keys to its children (leaves).

2. Leaf nodes are required to be in the range of their parent or the central authority to receive the new keys.

3. A child node who is out of the range of its parent, can receive its keys whenever it will come in the range of its parent or the central authority.

4. Since these nodes are mobile, some of them may stay in the range for very small time and therefore they have to receive their new keys in the given time only.

5. No node is allowed to receive keys of other nodes.

The set of new keys is transferred to the corresponding node using "data packets" over the specified internet protocol suite. An internet protocol suite is a collection of four layers. They are defined as application layer, transport layer, internet layer and link layer. We are using the internet protocol suite containing layers as File Transfer Protocol (FTP), Transmission Control Protocol (TCP), Internet Protocol (IP) and Medium Access Control (MAC-802_11) respectively.

Data packet - When a file is sent from one place to another on a network, the Transmission Control Protocol (TCP) layer divides the file into packets. Each of these packets is labelled along with an address of their destination. These labelled packets are called data packets. Such packets can follow different paths to reach their destination and once they all arrive to the other end, TCP reassembles them back into the original file.

Routing protocol - A routing protocol is used to determine optimal data transfer by defining specific communication paths between nodes of the network. It uses routing algorithms and distributing information which helps it to select optimized routes between any two nodes on a network. Ad-hoc On-Demand Vector Routing (AODV), Dynamic Source Routing (DSR) and Destination-Sequenced Distance-Vector Routing (DSDV) are some examples of MANET routing protocols. DSR uses a route-table of the source node to create new routes (when needed) for transmission of packets. For lower mobility models, DSR performs better than other routing protocols therefore we have used DSR as the routing protocol for our simulation.

5.3.2 Simulation Data

For this simulation, scripts are written using TCL (to generate trace files), Perl (to extract relevant data from the trace files) and Python (to organize data and plot graphs). The simulation is done using the network simulator NS-2 [17]. This simulation ran for 60 seconds with DSR routing protocol and returned the following data as shown in the Figure 5.2 and Table 5.8.

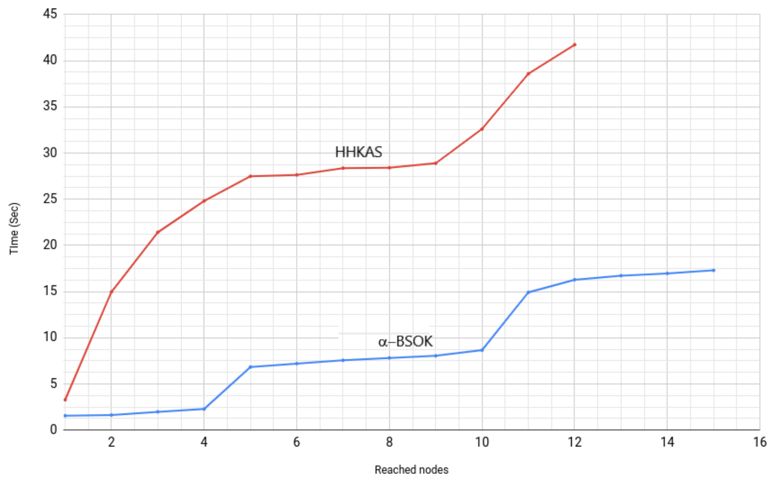

Fig. 5.2 HHKAS vs α-BSOK; Key distribution time with DSR routing.

	α-BSOK	HHKAS
Reached nodes (out of 15)	15 (all of them)	12
Time elapsed (out of 60 sec)	17.28	41.73

Packets information -

	Data packets ($\sim$ 572 bytes)	DSR packets ($\sim$ 48 bytes)
Total packets sent:	9295 (only AGT)	868 (only AGT)
Total packets received:	13437 (AGT + RTR)	2146 (AGT + RTR)
Total packets dropped:	126	41

Table 5.8 HHKAS vs α-BSOK (with DSR); Number of nodes which got their complete keys and the time taken till the last completed node.

In Figure 5.2, we can clearly see that during a key-refreshment, α-BSOK requires less time to distribute keys in the network than HHKAS. This behaviour can be explained from the difference of the key set sizes in α-BSOK and HHKAS. In our model, key set size for a leaf node is 55,040 bytes and 384 bytes for HHKAS and α-BSOK respectively. A typical packet size in MANET is 512 bytes. That means we need to transfer 108 packets to pass a complete key set to a child with HHKAS whereas with α-BSOK, this can be done within a single packet transfer.

Another observation which can be inferred from these data is that in the 60 seconds of the simulation, α-BSOK takes 17.28 seconds to distribute key sets to all 15 nodes whereas HHKAS takes 41.73 seconds to transfer key sets of only 12 nodes.

Remark - In our network, data packets contains information about the secret keys of destination node, therefore, packet routing is not always a good choice.

In 2004, Joerg Widmer proposed a no routing protocol named NOAH [31]. We have replaced DSR with NOAH in the same simulation to analyze the performance of α-BSOK with no routing involved. This simulation also ran for 60 seconds with no routing protocol and returned the following data as shown in the Figure 5.3 and Table 5.9.

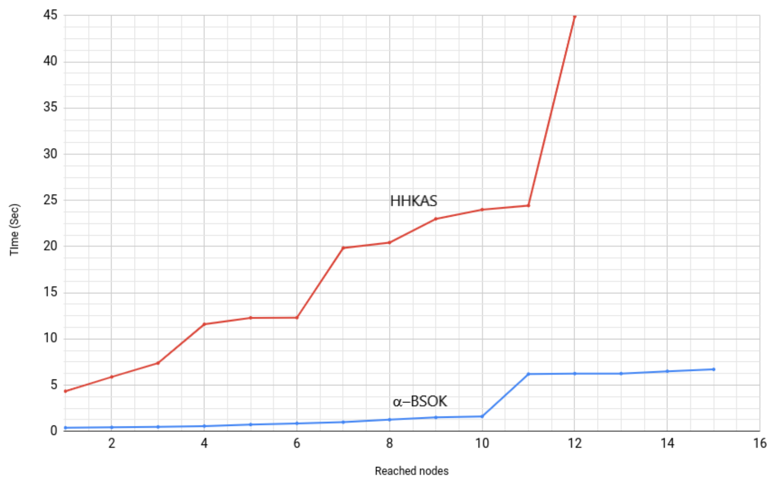

Fig. 5.3 HHKAS vs α-BSOK; Key distribution time without routing.

	α-BSOK	HHKAS
Reached nodes (out of 15)	15 (all of them)	12
Time elapsed (out of 60 sec)	6.72	45

Packets information -

	Data packets ($\sim$ 572 bytes)	Routed packets
Total packets sent:	10580	0
Total packets received:	10123	0
Total packets dropped:	422	0

Table 5.9 HHKAS vs α-BSOK (without routing); Number of nodes which got their complete keys and the time taken till the last completed node.

From Table 5.9, we can notice that when there is no routing, α-BSOK performs even better. In the 60 seconds of the simulation, α-BSOK takes 6.72 seconds to distribute key sets to all 15 nodes whereas HHKAS takes 45 seconds to transfer key sets of only 12 nodes.

In α-BSOK, due to smaller key set sizes, number of packets to be transferred is small per node and therefore a large number of nodes can get their keys at same time without the problem of high traffic loads or other issues of general sophisticated communications.
Remarks -

1. The simulation model given in HHKAS [16] uses the fact that during a key-refreshment, if a parent gets its new keys, it can instantly pass the corresponding new children-keys to its children and other descendants in range. Which means their model implicitly assumes that the time taken by a parent to generate children-keys from its own keys is negligible. But from the data shown in Table 5.4, we know that with HHKAS a parent requires very large amount of time to generate keys for its children. Whereas for α-BSOK, the time of key generation can be considered negligible than the time taken in transferring the packets. Hence we can say that in comparison to HHKAS, simulation results of α-BSOK are more reliable and close to the real scenario.

2. Time taken to transfer the packets does not only depend on key exchange protocol but also depend on network parameters, traffic loads and other factors like total time a node stays in the range of its ancestors during the key refreshment period.

5.4 α-BSOK vs β-BSOK

In Section 4.2, we have presented a variant of α-BSOK protocol named as β-BSOK to avoid the key escrow problem. In this section, we present an analytical comparison between α-BSOK and β-BSOK protocol to list out the pros and cons of β-BSOK protocol.

β-BSOK differs from α-BSOK by an additional component defined as "spy" nodes which are assumed to be non-malicious and uncompromising. Remember that minimum security provided by β-BSOK is always same as α-BSOK (i.e. secure under H-IND-SK) and if the spy nodes follow the protocol as described then β-BSOK can provide a stronger security.

From the descriptions of these protocols, we can notice that the key-storage and shared key computation time requirements of β-BSOK are identical to the corresponding requirements of α-BSOK because these parameters are invariant of the "spy" nodes. Whereas the key-generation time increases from α-BSOK to β-BSOK by a very small constant i.e., if in an ℓ-levelled hierarchy, a node at L^{th} level requires T_α time to generate keys for its $(L+1)^{th}$ level child using α-BSOK protocol then it will require $T_\alpha + (n_{\ell-1} \cdot \tau)$ time to generate keys for the same child node using β-BSOK where τ $(\sim 4.7 \times 10^{-7} sec)$ is the time required to do a multiplication in $\mathbb{Z}_q^*$ and n_i is the total number of nodes from first level to the i^{th} level of the hierarchy (note that time required to send and receive keys from a spy node is already counted in communication cost below and thus is not included in the key generation time). Since key distribution time depends upon key generation time and key-storage of the corresponding parent-child pair, therefore, we can say key distribution time of α-BSOK and β-BSOK will also be almost same.

Only cons so far of the β-BSOK protocol is that it requires additional communications with fertility carriers to generate remaining keys for a node and to add new children in the network. However, we would like to emphasize a point in β-BSOK that it is not necessary to separately communicate for each new node admission i.e., a parent can ask the corresponding fertility carrier to convert the keys for all of its new children in a single communication.

5.5 α-BSOK vs Trivial Approach

Using α-BSOK over the Trivial solution has certain gains and losses. In this section, we describe these pros and cons in details.
Pros :

1. **PKG workload** - From Section 3.5, we know that in Trivial solution, the root node is required to generate and store $\frac{N^2}{2}(1 - \frac{1}{N})$ number of keys in its storage where N represents the total number of nodes in the network. This creates a high workload on the root node (PKG). Whereas in α-BSOK, the same root node is required to generate and store only b_1 number of keys where b_1 represents the number of nodes in the first level.

2. **Flexibility** - Trivial solution is not flexible towards a direct node admission i.e., a parent can not generate valid secret-keys for a new child without the help of its ancestors (including the PKG). Whereas α-BSOK is flexible in both length wise and width wise (branching wise) direct admissions of new nodes.

3. **Communication cost** - In Trivial solution, all secret keys are generated only by the PKG. That means, an intermediate node can not generate the keys for any of

its descendants and therefore all the keys of a lower level node are required to be transferred from PKG to it through all of its ancestors. To exemplify, with Trivial solution in an ℓ-levelled hierarchy, providing the secret key set of n elements to a leaf node generally requires ℓn elements to be transferred throughout the hierarchy. However, in α-BSOK, this can be done with the communication cost of transferring only the n elements.

Cons :

1. **SHK computation time** - α-BSOK requires one pairing computation to generate the shared secret key between a pair of nodes whereas in the Trivial solution, no such computations are required. Which makes the Trivial solution a good choice for negligible computing powered devices but if we can bare the cost of one pairing computation then α-BSOK can be considered as a better option for H-ID-NIKE.

2. **Bit-size of key-storage** - Although in terms of keys, the key storage of a node in α-BSOK is always smaller in comparison to the corresponding key-storage in the Trivial solution but when it comes to the bit-size of the key storage, we can not make the same statement always. This is because the bit-size of a key depends upon the underlying security parameters which differ with protocols. Note that certain thresholds can be defined to ensure that the "bit-size" of a key set is smaller in α-BSOK than in the Trivial solution.

For example, in an ℓ-levelled hierarchy, we can define a threshold on the second last level's branching factor (B) as follows to ensure that the "bit-size" of a key set is smaller in α-BSOK in comparison to the Trivial solution -
let us consider $\mathcal{K}$, s and b respectively as the key set of a node, bit-size of a key in the key set and the minimum branching factor in the hierarchy then we want the following inequality to hold for all values of N,

$$\begin{aligned}
min[_{\textit{bit-size}}\{\mathcal{K}_{\text{Trivial}}\}] \geq\ & max[_{\textit{bit-size}}\{\mathcal{K}_{\alpha\text{-BSOK}}\}] \\
\implies \quad s_T \cdot [N-1] \geq\ & max\left[s_\alpha \cdot \left\{\frac{N}{1+(1-b^{-1})B}\right\}\right];\ b \geq 2 \\
\implies \quad s_T \cdot [N-1] \geq\ & s_\alpha \cdot \left[\frac{N}{1+B/2}\right] \\
\implies \qquad\qquad B \geq\ & 2\left[\frac{s_\alpha}{s_T} \cdot \left\{\frac{N}{N-1}\right\} - 1\right]. \qquad \cdots\cdots (1)
\end{aligned}$$

We can further simplify this by saying that if N takes moderate to large values then we can assume that -

$$\begin{aligned} N-1 &\geq \frac{s_\alpha}{s_T} \\ \implies 2 &\geq \frac{2s_\alpha}{s_T(N-1)} \\ \implies \frac{2s_\alpha}{s_T} &\geq 2\left[\frac{s_\alpha}{s_T}\cdot\left\{\frac{N}{N-1}\right\}-1\right]. \end{aligned}$$

Hence, in such cases, the following inequality can take care of (1) -

$$B \geq \frac{2s_\alpha}{s_T}.$$

That means, for a given hierarchy, if B satisfies this inequality then we can say that α-BSOK is a better choice for that hierarchy in terms of key storage also.

5.6 α-BSOK and β-BSOK vs Existing Solutions

Apart from HHKAS, we have also studied and analyzed other existing protocols which were defined for the fulfillment of the same purpose i.e. key exchange. In Table 5.10, we present some comparisons of α-BSOK and β-BSOK with these protocols on different important features including flexibility towards new node admission, computational efficiency and leaf storage requirement for full resilience. Notations used in Table 5.10 are described as follows :

N -	Total number of nodes in the network
NA -	Feature not applicable
ℓ -	Number of levels in the hierarchy
b -	Minimum branching factor up to level $\ell-2$ with $b \geq 2$
b_1 -	Number of nodes at level 1
B -	Branching factor at level $\ell-1$
W -	Protocol is flexible with new node admissions at any level of the hierarchy
L -	Protocol is flexible with a new level admission in the hierarchy
e -	Euler's number
✓ -	Protocol contains the feature
✗ -	Protocol does not contain the feature
LD -	A left most descendant of relatively higher levelled target node.
m -	Security parameter required to ensure $(m\log_2 e)$-bit security
h -	Hash computation into $\mathbb{Z}_q^*$
a -	Addition in $\mathbb{Z}_q^*$
p -	Multiplication in $\mathbb{Z}_q^*$
H -	Hash computation into $\mathbb{G}$
A -	Addition in $\mathbb{G}$
P -	Pairing into $\mathbb{G}_T$
M -	Secure multilinear mapping into some $\mathbb{G}'_T$ with $\lvert\mathbb{G}'_T\rvert = q$

Protocols $\rightarrow$	BIOS[20]	SOK[25]	Freire[12]	Pert.poly[29]	HHKAS[16]	BIOS-SOK[27]	α-BSOK	β-BSOK	Trivial
Hierarchical	✗	✗	✓	✓	✓	✓	✓	✓	✓
ID-based	✓	✓	✓	✓	✓	✓	✓	✓	✓
Non-interactive	✓	✓	✓	✓	✓	✓	✓	✓	✓
Resiliency	Fully	Fully	Fully	t-threshold	t-threshold	LD-threshold	Fully	Fully	Fully
HINDSK-secure [3.2]	NA	✓	✓	✗	✗	✗	✓	✓	✓
Key-escrow free [4.1.3]	✗	✗	✗	✗	✗	✗	✗	✓	✗
Intra-level $\mathcal{SHK}$ [4.1.1]	✓	✓	✓	✓	Leaf-level only	✓	✓	✓	✓
Inter-level $\mathcal{SHK}$ [4.1.1]	NA	NA	✓	✓	✗	✓	✓	✓	✓
Flexibility	W	W	$W + L$	W	W	$W + L$	$W + L$	$W + L$	✗
Maximum required shared-key op^{n}s	$1h$	$1H$ $+1P$	$1H$ $+1M$	$2\ell h + [^{2\ell+t}C_{2\ell} - 1]a$ $+[(\frac{2t\ell}{2\ell+1})^{2\ell+t}C_{2\ell}]p$	$[\frac{2m}{t}(et)^{\ell-1}(1+t)^{\ell}]h + 1H$ $+[m(et)^{\ell-1}]A + 1P$	$1h + 1H$ $+[\ell - 1]P$	$1H$ $+1P$	$1H$ $+1P$	*Negligible*
Root key-storage	N	1	1	$^{2\ell+t}C_{2\ell}$	$m[et(1+t)]^{\ell-1} + 2\log N$	b_1	b_1	$b_1 + 1$	$\frac{N^2}{2}(1 - \frac{1}{N})$
Key-storage at level $i \neq \ell$	–	–	1	$^{2\ell-i+t}C_{2\ell-i}$	$\frac{m[et(1+t)]^{\ell-1}}{(1+t)^i} + \frac{2\log N}{(1+t)^i}$	$< \frac{b_1}{2} + \left[\frac{N+2}{(1-b^{-1})^{-1}+B}\right]^{\frac{i-1}{\ell-1}}$	$< \frac{N}{1+(1-b^{-1})B}$	$< \frac{N}{1+(1-b^{-1})B}$	$\frac{N^2}{b^i}\left[1 - \frac{1}{N^{(\ell-i+1)/\ell}}\right]$
Leaf key-storage	$\lceil\frac{N}{2}\rceil$	1	1	$^{\ell+t}C_{\ell}$	$m(et)^{\ell-1} + \frac{2\log N}{(1+t)^{\ell-1}}$	$< \frac{N+2}{(1-b^{-1})^{-1}+B}$	$< \frac{N}{1+(1-b^{-1})B}$	$< \frac{N}{1+(1-b^{-1})B}$	$N - 1$
Leaf key-sto. for full resi.	$\lceil\frac{N}{2}\rceil$	1	1	$> \frac{N^2}{4(1+B^{-1})^2}$	$> me^{\ell-1}\left[\frac{N+2}{(1-b^{-1})^{-1}+B}\right]^{\ell/2}$	–	$< \frac{N}{1+(1-b^{-1})B}$	$< \frac{N}{1+(1-b^{-1})B}$	$N - 1$
Comp. efficiency	Medium	High	Impractical	Medium	Low	Medium	Medium	Medium	Medium

Table 5.10 α-BSOK and β-BSOK vs Existing Solutions.

From the Table 5.10, we can say that all over α-BSOK (or β-BSOK; depending upon the application) is the best available choice for key-exchange in resource-constraint networks.

5.7 Summary

We have presented a comparative analysis of α-BSOK protocol with other existing protocols. From these implementation data, simulation data and the data presented in the Table 5.10, we can conclude that α-BSOK is the only available practical protocol which is flexible and secure under HINDSK model and not only provides the four functional properties posed by Gennaro *et al.*[16] but also allows multi-level shared-key computations.

5.8 Limitations of α-BSOK Protocol

Some of the observed limitations of α-BSOK are described as follows :

1. For an ℓ-levelled hierarchy, the key-storage of a node in α-BSOK is polynomially dependant upon the branching factors up to level $\ell - 2$. In particular, adding a non-leaf node in the network increases the key storage of each node by one key. On the other hand, this key storage is invariant of any increase in leaf nodes (i.e., the branching factor at level $\ell - 1$). Therefore, α-BSOK may perform poorly for large hierarchies which contain most of its nodes in non-leaf levels. However, we believe in most of the real life applications such as in military, α-BSOK may be adequate. This is because usually in such applications branching factors up to level $\ell - 2$ are modest and comparatively small than the branching factor at level $\ell - 1$.

2. Key storage in α-BSOK depends exponentially upon the hierarchical depth. In particular, the number of keys per node gets multiplied by the branching factor of the previous second last level as we add a new level in the hierarchy.

Thus α-BSOK can only be a suitable choice for hierarchies with moderate hierarchical depths and moderate branching factors up to level $\ell - 2$.

Chapter 6

Conclusion

Key exchange is a fundamental functionality for secure communication. Using a key exchange protocol, any pair of nodes can agree on a shared-key which can be used to protect their secret communications over insecure channels. Furthermore, in real life applications such as in military, we also desire the key exchange protocol to be decentralized, non-interactive, flexible, efficient and fully resilient. Thus a Hierarchical Identity-based Non-interactive Key Exchange (H-ID-NIKE) with full resilience and flexible structure can be a perfect solution for such applications.

In this thesis, we have proposed α-BSOK as a H-ID-NIKE protocol which contains all the properties mentioned above. From the data analysis and comparisons shown in Chapter 5, we can conclude that α-BSOK is the best available, practical solution for real life hierarchical networks with moderate depth and branching factors. We also claim that α-BSOK is the only practical key exchange protocol which not only provides the four functional properties presented in [16] but also supports multi-level shared-key computations and new node admissions. The security guarantee of α-BSOK is claimed with a rigorous proof of security under a more practical and stronger security model than the existing protocols [27, 16, 12].

Some of the take away open problems from this thesis are described as follows :

1. In α-BSOK, key storage of a node directly depends upon the two parameters - hierarchical depth and branching factor. Thus it is still an open problem to construct an H-ID-NIKE protocol which is not only secure and efficient but can also easily accommodate arbitrary number of levels with no branching limit i.e., the storage requirements are independent of either or both of these parameters.

2. α-BSOK is shown adaptive secure under the random oracle model. However, it is still an open problem to construct an H-ID-NIKE protocol which can be shown adaptive secure in the standard model i.e., without assuming the existence of random oracle (see Section 2.5.1) .

References

[1] Akinyele, J. A., Garman, C., Miers, I., Pagano, M. W., Rushanan, M., Green, M., and Rubin, A. D. (2013). Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128.

[2] Bellare, M. (1997). Practice-oriented provable-security. In *Information Security, First International Workshop, ISW '97, Tatsunokuchi, Japan, September 17-19, 1997, Proceedings*, pages 221–231.

[3] Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73.

[4] Bellare, M. and Rogaway, P. (1996). The exact security of digital signatures - how to sign with RSA and rabin. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 399–416.

[5] Blundo, C., Santis, A. D., Herzberg, A., Kutten, S., Vaccaro, U., and Yung, M. (1998). Perfectly secure key distribution for dynamic conferences. *Inf. Comput.*, 146(1):1–23.

[6] Boneh, D. and Franklin, M. K. (2001). Identity based encryption from the weil pairing. *IACR Cryptology ePrint Archive*, 2001:90.

[7] Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pages 514–532.

[8] Coron, J., Lee, M. S., Lepoint, T., and Tibouchi, M. (2016). Cryptanalysis of GGH15 multilinear maps. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 607–628.

[9] Coron, J.-S. (2000). On the exact security of full domain hash. In *Annual International Cryptology Conference*, pages 229–235. Springer.

[10] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654.

[11] Eschenauer, L. and Gligor, V. D. (2002). A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 41–47.

[12] Freire, E. S. V. (2014). *Non-interactive key exchange and key assignment schemes*. PhD thesis, Stanford University, California.

[13] Frey, G., Müller, M., and Rück, H. (1999). The tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *IEEE Trans. Information Theory*, 45(5):1717–1719.

[14] Galbraith, S. D. (2005). The weil pairing on elliptic curves over C. *IACR Cryptology ePrint Archive*, 2005:323.

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

[16] Gennaro, R., Halevi, S., Krawczyk, H., Rabin, T., Reidt, S., and Wolthusen, S. D. (2008). Strongly-resilient and non-interactive hierarchical key-agreement in manets. In *European Symposium on Research in Computer Security*, pages 49–65. Springer.

[17] Issariyakul, T. and Hossain, E. (2008). *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1 edition.

[18] Joux, A. (2000). A one round protocol for tripartite diffie-hellman. In *Algorithmic Number Theory, 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings*, pages 385–394.

[19] Katz, J. and Wang, N. (2003). Efficiency improvements for signature schemes with tight security reductions. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 155–164.

[20] Lee, J. and Stinson, D. R. (2004). Deterministic key predistribution schemes for distributed sensor networks. In *International Workshop on Selected Areas in Cryptography*, pages 294–307. Springer.

[21] Lynn, B. (2007). *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University, California.

[22] NIST (2002). Secure hash standard. *Federal Information Processing Standards Publication 180-4*.

[23] Paterson, K. G. and Srinivasan, S. (2009). On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. *Designs, Codes and Cryptography*, 52(2):219–241.

[24] Ramkumar, M., Memon, N., and Simha, R. (2005). A hierarchical key predistribution scheme. In *Electro Information Technology, 2005 IEEE International Conference on*, pages 6–pp. IEEE.

[25] Sakai, R., Ohgishi, K., and Kasahara, M. (2000). Cryptosystems based on pairing. *The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 45:26-28.*

[26] Shamir, A. (1984). Identity-based cryptosystems and signature schemes. In *CRYPTO*, LNCS, pages 47–53. Springer.

[27] Tiwari, M. (2017). *Fully Resilient Non-Interactive ID-Based Hierarchical Key Agreement*. Master's thesis, Indian Institute of Science, Bangalore.

[28] Toh, C. K. (2001). *Ad hoc mobile wireless networks: protocols and systems*. Pearson Education.

[29] Wang, Q., Khurana, H., and Nahrstedt, K. (2009). Non-interactive hierarchical pairwise key predistribution scheme with multi-level key establishment.

[30] Weil, A. (1940). Sur les fonctions algébriquesa corps de constantes fini. *CR Acad. Sci. Paris*, 210(1940):592–594.

[31] Widmer, J. (2004). *NO Ad-Hoc routing agent (NOAH)*. Swiss Federal Institute of Technology - Lausanne.

Appendix A

Tree Structure

In the Section 5.1, we have defined a class of hierarchies $\mathcal{H}$. Any tree structure from this class can be used to do a fair efficiency comparison between fully resilient HHKAS and α-BSOK. For our analysis, We have used a 5-levelled hierarchy $H \in \mathcal{H}$. The number of nodes N_i in the corresponding level i of H is defined as -

$$N_1 = 9, N_2 = 17, N_3 = 33, N_4 = 65 \text{ and } N_5 = 129.$$

The hierarchical positions of the nodes in H are shown (with labels and ranks) in the following figure.