Secure Proxy Signature Schemes for Delegation of Signing Rights

ALEXANDRA BOLDYREVA [*]   ADRIANA PALACIO [†]   BOGDAN WARINSCHI [‡]

Abstract

A proxy signature scheme permits an entity to delegate its signing rights to another. These schemes have been suggested for use in numerous applications, particularly in distributed computing. Before our work [6] appeared, no precise definitions or proven-secure schemes had been provided. In this paper, we formalize a notion of security for proxy signature schemes and present provably-secure schemes. We analyze the security of the well-known delegation-by-certificate scheme and show that after some slight but important modifications, the resulting scheme is secure, assuming the underlying standard signature scheme is secure. We then show that employment of aggregate signature schemes permits bandwidth and computational savings. Finally, we analyze the proxy signature scheme of Kim, Park and Won, which offers important performance benefits. We propose modifications to this scheme which preserve its efficiency and yield a proxy signature scheme that is provably secure in the random-oracle model, under the discrete-logarithm assumption.

Keywords: Digital signatures, proxy signatures, aggregate signatures, provable security.

1 Introduction

A proxy signature protocol allows an entity, called the *designator* or *original signer*, to delegate another entity, called a *proxy signer*, to sign messages on its behalf, in case of say, temporal absence, lack of time or computational power, etc. The delegated proxy signer can compute a *proxy signature* that can be verified by anyone with access to the original signer's certified public key. We note that Blaze and Strauss[5] and Dodis and Ivan [15] use the term "proxy signatures," in the context of "proxy cryptography," to describe a different primitive with distinct goals.

APPLICATIONS AND BACKGROUND. Proxy signatures have found numerous practical applications, particularly in distributed computing where delegation of rights is quite common. Examples discussed in the literature include distributed systems [31, 43], grid computing [10], mobile agent applications [16, 20], distributed shared object systems [23], global distribution networks [1], and mobile communications [33]. The proxy signature primitive and the first efficient solution were introduced by Mambo, Usuda and Okamoto [29]. Since then proxy signature schemes have enjoyed a considerable amount of interest from the cryptographic research community. New security considerations and constructions have been proposed, old schemes have been broken, followed by more constructions (e.g., [17, 46, 32, 40, 39, 47, 20, 11, 21, 41, 44, 9, 25, 48, 45]). Furthermore, many extensions of the basic proxy signature primitive have been considered. These include threshold proxy signatures [17, 50, 37, 38, 14, 13], blind proxy signatures [18, 48], proxy signatures with warrant recovery [19], nominative proxy signatures [33], one-time proxy signatures [16], and proxy-anonymous proxy signatures [36].

Unfortunately, the extensive previous cryptographic research on the topic has not brought developers much guidance because almost every other paper breaks some previously proposed construction, and proposes a new

[*] College of Computing, Georgia Institute of Technology, USA E-Mail: aboldyre@cc.gatech.edu.

[†] Computer Science Department, Bowdoin College, USA E-Mail: apalacio@bowdoin.edu.

[‡] Computer Science Department, University of Bristol, UK. E-Mail: bogdan@cs.bris.ac.uk.

one. See [49, 22, 20, 21, 41] for illustrative examples of this trial and error approach. Very few schemes were left unbroken, and none of them had provable-security guarantees. Typically, security of these schemes is argued by presenting attacks that fail, which provides only very weak guarantees. What is clearly desirable but has not been provided until now, is a proxy signature scheme with *guaranteed* security. In order to achieve this goal, it is necessary to first formalize a security notion for proxy signature schemes, since the current security requirements are vague and ill-defined. This problem was recognized and left open in [21].

Our work is aimed at filling this void. The original version of the paper [6] is the *first* work on proxy signatures in the provable-security direction. We define a formal model for the security of proxy signature schemes, which enables the cryptographic analysis of such schemes. Then we present several examples of efficient proxy signature schemes that *provably* satisfy this notion of security, under widely-believed computational-complexity assumptions.

We note that the contribution of our work is not only in our immediate results, but also in its impact on further research on proxy-signature-related topics under the principles of provable security, as exemplified by [13, 28]. Herranz and Saez [13] extend our security model to analyze fully distributed proxy signatures. Building on our work, Malkin et al. [28] give a model for hierarchical proxy signatures and investigate foundational issues such as their relation with several key-evolving signature primitives.

FUNCTIONALITY AND SECURITY OF PROXY SIGNATURE SCHEMES. As in previous works, we assume a Public Key Infrastructure (PKI) setting, where each entity holds a public and secret key pair. As usual, each user can sign messages using the signing algorithm of a standard digital signature scheme, and his or her secret key. When a user (the original signer) desires to delegate his or her signing ability to another user (the proxy signer), the users run a possibly interactive *proxy-designation* protocol. We note that a proxy signer can correspond to another device (e.g., a palm computer) of the original signer. Through a successful execution of this protocol, the proxy signer obtains a proxy signing key. It can then sign messages on behalf of the original signer using a *proxy signing* algorithm and the proxy signing key. Anyone can verify the validity of such signatures using a *proxy verification* algorithm and the original signer's public key.

Several security properties for proxy signature schemes were introduced in [29], were somewhat enhanced by [20], and did not evolve much since then. The properties stated in [20] are the following.

Verifiability: From a proxy signature, a verifier can be convinced of the original signer's agreement on the signed message.

Strong unforgeability: The original signer and third parties who are not designated as proxy signers cannot create a valid proxy signature.

Strong identifiability: Anyone can determine the identity of the corresponding proxy signer from a proxy signature.

Strong undeniability: A proxy signer cannot repudiate a proxy signature it created.

Prevention of misuse: A proxy signing key cannot be used for purposes other than generating valid proxy signatures. In case of misuse, the responsibility of the proxy signer should be determined explicitly.

While these informal requirements provide some intuition about the goals that a notion of security for proxy signature schemes should capture, their precise meaning is unclear. They do not specify what a successful attack is, leaving important questions unanswered. For instance, what are an adversary's capabilities? In particular, can malicious parties collude? Are attackers allowed to see or request signatures? Are they allowed to register keys? What exactly is the adversary's goal? When is an attacker considered successful?

One of the main contributions of our work is to clarify these issues by designing a formal model for the security of proxy signature schemes. It involves a rather powerful adversary who is allowed to corrupt an arbitrary number of users and learn their secret keys. Moreover, the adversary can register public keys on behalf of new users, possibly obtained otherwise than running the key-generation algorithm, and possibly depending on the public keys of already registered users.

We allow the adversary to interact with honest users playing the role of a designator or that of a proxy signer. We also allow it to see transcripts of all executions of the proxy-designation protocol between honest users, i.e., we do not assume the existence of a secure channel between a designator and a proxy signer. The adversary can ask to see both standard signatures and proxy signatures generated by honest users on messages of her choice. We say that the adversary wins if it manages to create a standard signature or a proxy signature for a new message, i.e., a message that was not signed by an honest user. We say that a proxy signature scheme is secure if no probabilistic efficient adversary can win with probability non-negligible in the security parameter of the scheme. This security model is detailed in Section 3.

CONSTRUCTIONS. The simplest approach to achieve the main goal of a proxy signature scheme is for the designator to give its secret key to the proxy signer, who can then use it to sign messages. In this case proxy signatures are just standard signatures, and can be verified the usual way. This scheme, called *full delegation* in the literature, has several shortcomings. Its security relies on the honesty of the proxy signer in a completely unrealistic manner. It provides no way to restrict signing rights to particular types of messages or a certain time period. Even if the proxy signer is fully trusted, this scheme increases the vulnerability of the designator's secret key. Additionally, it requires the establishment of a secure channel between the original signer and the proxy signer. Although most previous works assume a secure channel for the proxy-designation protocol, we find this requirement unnecessary and undesirable.

Another simple construction is known as *delegation by certificate* or *delegation by warrant*. Here the designator uses the signing algorithm of a standard signature scheme to produce an unforgeable *warrant* that certifies that the proxy signer is indeed allowed to sign on its behalf. Usually, the warrant consists of a description of the space of messages for which the proxy signer is allowed to produce signatures, together with a signature on this description (and possibly some other information like the identity of the designator, and/or that of the proxy signer). We refer to this signature as a *certificate*.

The warrant is sent to the proxy signer who uses it in conjunction with its own signing key to produce proxy signatures. A proxy signature contains the warrant and the proxy signer's signature. A verifier needs to ensure that the certificate contained in the warrant is valid with respect to the public key of the designator, verify the second signature with respect to the public key of the proxy signer specified in the warrant, and also ensure that the message signed belongs to the message space specified in the warrant. One of our contributions is to show that a direct implementation of this scheme is susceptible to a chosen-message attack which we present in Section 4. We also discuss other vulnerabilities of naive implementations, including one pointed out by [28]. We provide fixes and prove that the resulting scheme is secure, assuming the underlying standard signature scheme is secure (i.e., existentially unforgeable under adaptive chosen message attack [12]). Also, we investigate other vulnerabilities of naive implementations, including the ones pointed out by [28] and [42].

A delegation-by-certificate proxy signature can be computed in roughly the same amount of time required for standard signing, but verification of such proxy signatures requires twice the time to verify a standard signature. Most of the works on basic proxy signatures mentioned above focused on constructing a more efficient scheme, where verification of a proxy signature requires less time than verification of two standard signatures. Several such constructions were proposed, but they all lack provable-security guarantees.

Aggregate signature schemes [7, 26, 24, 27] allow composition of a single short signature out of signatures generated by several users for different messages. Using an aggregate signature scheme, it is possible to obtain an improvement over delegation-by-certificate proxy signature schemes in terms of both bandwidth and efficiency. In Section 5 we discuss a simple construction of a secure proxy signature scheme, given any secure aggregate signature scheme, such that a proxy signature consists of, essentially, a message space description and a single aggregated signature. We prove that the resulting proxy signature scheme is secure under the assumptions needed for security of the base aggregate signature scheme.

For example, if a proxy signature scheme is based on the bilinear co-GDH aggregate signature scheme of Boneh et al. [7], the length of a proxy signature is the length of the message space description plus the length of a *single* short bilinear co-GDH aggregate signature. Verification of a proxy signature requires three bilinear map computations, whereas verification of a delegation-by-certificate proxy signature based on the bilinear co-GDH signature scheme requires four bilinear map computations. The bilinear co-GDH aggregate signature scheme was proved secure in the random-oracle (RO) model [4], under the Computational co-Diffie-Hellman assumption[1]. Thus the corresponding proxy signature scheme is secure under the same assumption in the RO model.

Kim, Park, and Won [17] presented a proxy signature scheme (KPW) based on the Schnorr signature scheme [35] that is more efficient than the delegation-by-certificate scheme based on Schnorr signatures. The latter requires four modular exponentiations per proxy signature verification, while verification of a KPW proxy signature requires only three exponentiations. This improvement can be furthered by employing an algorithm for simultaneous multiple exponentiation (e.g., Algorithm 14.88 in [30]). The cost of verification of a KPW proxy signature can be reduced to about 1.25 exponentiations, versus roughly 2.5 exponentiations for verification of a Schnorr-based delegation-by-certificate proxy signature using standard techniques for fast exponentiation. To the best of our knowledge, the KPW scheme is the only proxy signature scheme that remained unbroken before the earlier version of our work appeared, does not require a secure channel for proxy designation, and has this advantage in efficiency over the corresponding delegation-by-certificate scheme.

We discuss the KPW scheme in Section 6. We mention that we were unable to prove the original scheme secure. We modify the scheme, preserving its efficiency, and prove that the resulting scheme is secure in the random-oracle model, assuming hardness of computation of discrete logarithms in the underlying group. Our proof is in the basic key-registration model, i.e., it does not assume that users prove knowledge of secret keys during public-key registration. We call this scheme the Triple Schnorr proxy signature scheme since it uses Schnorr signatures for standard signing, proxy designation, and proxy signing. The proof, which is our second main result, is quite technical and long, and we present it in Appendix C. Triple Schnorr is the first provably-secure proxy signature scheme with an advantage in efficiency of verification of about 50% over the corresponding delegation-by-certificate scheme. Standard signing and proxy signing require approximately the same amount of time in both of these schemes.

We note that our proof of security of the KPW scheme uses a new lemma of independent interest, that we call the Multiple-Forking Lemma. Our lemma is a farther generalization of the General Forking Lemma of Bellare and Neven [2].

Relation to the previous version of the paper [6]. This work significantly strengthens the earlier version in several ways. We extend the security model to address self-delegation (an issue raised by [28]) and warrants. We also give a new proof for the security of Triple Schnorr construction that does not require the users to prove knowledge of their secret keys during public key registration. We give concrete security results for our constructions.

2 Preliminaries

Notation. For $N \in \mathbb{N}$, we let $[N] = \{1, \ldots, N\}$. If A is a randomized algorithm, then the notation $x \overset{\$}{\leftarrow} A(x_1, x_2, \ldots)$ denotes that x is assigned the outcome of the experiment of running A on inputs $x_1, x_2, \ldots$ with fresh coins. If A is deterministic, we might drop the dollar sign above the arrow. For strings $a_1, \ldots, a_n$, $a_1 || \cdots || a_n$ denotes an encoding such that the constituent strings are uniquely recoverable from the final one. A (possibly randomized) algorithm is called efficient if it runs in time polynomial in the input length (which is usually the security parameter). A function $f\colon \mathbb{N} \to [0, 1]$ is called *negligible* if it approaches zero faster than the reciprocal of any polynomial, i.e., for any polynomial p, there exists $n_p \in \mathbb{N}$ such that for all $n \geq n_p$, $f(n) \leq 1/p(n)$.

Signature schemes. We recall the definitions of a digital signature scheme and its security. For simplicity we give all definitions in the standard model. To extend these definitions to the random-oracle model, all

[1] See [8] for the definition of this assumption.

algorithms including the adversary are given oracle access to one or more random functions $G, H, \ldots$, drawn from the set of all functions with appropriate domains and ranges.

Definition 2.1 [Digital signature scheme] A digital signature scheme $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ is specified by four efficient algorithms with the following functionalities.

- The randomized *parameter-generation* algorithm $\mathcal{G}$ takes input 1^κ, where κ is the security parameter, and outputs some global parameters $params$. These may contain, for example, a security parameter, the description of a cyclic group and a generator, and the description of a hash function. We assume that these parameters become publicly available.
- The randomized *key-generation* algorithm $\mathcal{K}$ takes input global parameters $params$ and outputs a pair (pk, sk) consisting of a public key and a matching secret key respectively.
- The (possibly) randomized *signing* algorithm $\mathcal{S}$ takes input a secret key sk and a message $M \in \{0,1\}^*$, and outputs a signature σ.
- The deterministic *verification* algorithm $\mathcal{V}$ takes input a public key pk, a message M and a candidate signature σ for M, and outputs a bit. We say that σ is a *valid* signature for M relative to pk if $\mathcal{V}(pk, M, \sigma) = 1$.

For any pair of keys (pk, sk) that can be output by $\mathcal{K}$ and any $M \in \{0,1\}^*$, it is required that $\mathcal{V}(pk, M, \mathcal{S}(sk, M)) = 1$ with probability one.

Definition 2.2 [Security of a digital signature scheme] Let $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme. Consider an adversary $\boldsymbol{A}$ that is given input a public key pk and access to a signing oracle $\mathcal{O}_{\mathcal{S}}(sk, \cdot)$, where pk and sk are matching keys generated via $params \xleftarrow{\$} \mathcal{G}(1^\kappa)$; $(pk, sk) \xleftarrow{\$} \mathcal{K}(params)$. The oracle takes input a message M and returns a signature $\sigma \xleftarrow{\$} \mathcal{S}(sk, M)$. $\boldsymbol{A}$ queries this oracle on messages of its choice, and eventually outputs a forgery (M, σ). The advantage of adversary $\boldsymbol{A}$ in attacking the scheme DS, $\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{A}}(\kappa)$, is the probability that σ is a valid signature on M relative to pk, and this message was not queried to the signing oracle. The probability is taken over all the random coins used in the experiment above. DS is said to be *secure against existential forgery under adaptive chosen-message attack (or, simply, secure)* if $\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{A}}(\kappa)$ is negligible. Here and for other definitions in the paper we adopt the convention that the *time complexity* of adversary $\boldsymbol{A}$ is the execution time of the entire experiment, including the time taken for parameter and key generation, and computation of answers to oracle queries.

MESSAGE SPACE DESCRIPTION. A message space descriptor ω_S for message space $S \subset \{0,1\}^*$ is a deterministic polynomial time Turing machine that computes the characteristic function of S. Throughout the paper we use S and ω_S interchangeably, so by slight abuse of notation we write $M \in \omega_S$ to indicate that $\omega_S(M) = 1$, or equivalently, that $M \in S$. Also, we use standard set operations with message space descriptors as operands.

3 Proxy Signature Schemes

THE SETTING. As discussed in the Introduction, we consider a PKI-like setting: users are identified by natural numbers, and we let pk_i denote the public key of user $i \in \mathbb{N}$, and sk_i denote the corresponding secret key.

3.1 Syntax of Proxy Signature Schemes

A proxy signature scheme involves a digital signature scheme for standard signing, a protocol that users run in order for one of them to designate the other as a proxy signer, a signing algorithm to be used by proxy signers (which can differ from the one used for standard signing), and a corresponding verification algorithm for proxy signatures. Additionally, the strong identifiability property mentioned in the Introduction suggests that there be an algorithm that extracts the identity of the proxy signer from a proxy signature. This identity is the natural number identifying the user. We note that identities of an original signer and its proxy can coincide in case of self-delegation. The definition we give uses message space descriptors (Section 2) to specify the space of messages for which proxy signers are allowed to produce signatures.

The following definition details the components of a proxy signature scheme.

Definition 3.1 [Proxy signature scheme] A *proxy signature scheme* is a tuple $\mathsf{PS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$, where the constituent algorithms run in polynomial time, $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ is a digital signature scheme, and the other components are defined as follows.

- $(\mathcal{D}, \mathcal{P})$ is a pair of interactive randomized algorithms forming the (two-party) *proxy-designation protocol*. The input to each algorithm includes two public keys pk_i, pk_j for the *designator* i and the *proxy signer* j, respectively. $\mathcal{D}$ also takes as input the secret key sk_i of the designator, the identity j of the proxy signer, and a message space descriptor ω for which user i wants to delegate its signing rights to user j. $\mathcal{P}$ also takes as input the secret key sk_j of the proxy signer. As a result of the interaction, the expected local output of $\mathcal{P}$ is skp, a *proxy signing key* that user j uses to produce proxy signatures on behalf of user i, for messages in ω. $\mathcal{D}$ has no local output. We write $skp \xleftarrow{\$} [\mathcal{D}(pk_i, sk_i, j, pk_j, \omega), \mathcal{P}(pk_j, sk_j, pk_i)]$ for the result of this interaction.
- $\mathcal{PS}$ is the (possibly) randomized *proxy signing* algorithm. It takes input a proxy signing key skp and a message $M \in \{0,1\}^*$, and outputs a proxy signature $p\sigma$.
- $\mathcal{PV}$ is the deterministic *proxy verification* algorithm. It takes input a public key pk, a message $M \in \{0,1\}^*$ and a proxy signature $p\sigma$, and outputs 0 or 1. In the latter case, we say that $p\sigma$ is a *valid* proxy signature for M relative to pk.
- $\mathcal{ID}$ is the *proxy identification* algorithm. It takes input a valid proxy signature $p\sigma$, and outputs an identity $i \in \mathbb{N}$ or $\perp$ in case of an error.

CORRECTNESS. We require that for any message space $\omega \subseteq \{0,1\}^*$ and for all users $i, j \in \mathbb{N}$, if skp is a proxy signing key for user j on behalf of user i for message space ω, i.e., $skp \xleftarrow{\$} [\mathcal{D}(pk_i, sk_i, j, pk_j, \omega), \mathcal{P}(pk_j, sk_j, pk_i)]$, then for every $M \in \omega$, $\mathcal{PV}(pk_i, M, \mathcal{PS}(skp, M)) = 1$ and $\mathcal{ID}(\mathcal{PS}(skp, M)) = j$ with probability one. Informally, this means that signatures produced with proxy signing keys are valid relative to the public key of the designator, and that the identity of the proxy signer can be extracted from proxy signatures that it produces.

3.2 A Notion of Security for Proxy Signature Schemes

SOME INTUITION. We consider a multi-party setting where parties have public/secret keys registered with some public authority. The adversary may corrupt users and learn their secret keys. The adversary can also add new users and register public keys for them. These keys do not have be distributed according to the distribution defined by the key-generation algorithm and, in principle, they can depend on, the public keys of honest users.

We focus on a seemingly extreme case in which the adversary is working against a *single* honest user, say user 1, and can select and register keys for all other users. Notice that this is without loss of generality since any attack that can be carried out in the presence of more honest users can be performed by having some of the users under the adversary's control behave honestly. The adversary can play the role of user $i \neq 1$ in executions of the proxy-designation protocol with user 1, as designator or as proxy signer. In both cases, the adversary may behave dishonestly in an attempt to obtain information from the honest user, and the adversary can actually decide for which message space the designation takes place. There is no restriction on the number of executions of the proxy-designation protocol between the same two users. To account for the possibility that a user may designate itself as a proxy signer (which is useful, for example, to create a temporary key for use in a hostile environment), we let the adversary request user 1 to run the proxy-designation protocol with itself, and see the transcript of the execution. We emphasize that we do not assume the existence of a secure channel between a designator and a proxy signer. As pointed out by Malkin et al. [28], the adversary should also be allowed to obtain the proxy signing keys produced when user 1 designates itself since self-delegation is often employed in situations where the signing key is vulnerable to exposure.

We model chosen-message attack capabilities by providing the adversary access to two oracles: a standard signing oracle and a proxy signing oracle. The first oracle takes input a message M, and returns a standard signature for M by user 1. The second oracle takes input a tuple (i, l, M), and, if user 1 was designated by user i at least l times, returns a proxy signature for M created by user 1 on behalf of user i, using the l-th proxy signing key.

The goal of the adversary is to produce one of the following forgeries:

1. a standard signature by user 1 for a message that was not submitted to the standard signing oracle,
2. a proxy signature for a message M by user 1 on behalf of some user $i \neq 1$ such that any query (i, l, M) made to the proxy signing oracle was answered with $\perp$.
3. a proxy signature for a message M by user 1 on behalf of user 1 such that any query $(1, l, M)$ made to the proxy signing oracle was answered with $\perp$, and the adversary did not compromise any proxy signing key produced during self-delegation for a message space to which M belongs.
4. a proxy signature for a message M by some user $i \neq 1$ on behalf of user 1 such that user i was never designated by user 1 as a signer for a message space to which M belongs.

Our notion of security for proxy signature schemes is formally defined as follows.

Definition 3.2 [Security of a proxy signature scheme] Let $\mathsf{PS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$ be a proxy signature scheme, $\boldsymbol{A}$ an adversary and $\kappa \in \mathbb{N}$. We associate to PS, $\boldsymbol{A}$ and κ an experiment $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{PS},\boldsymbol{A}}(\kappa)$. First, system parameters $params$ are generated by running $\mathcal{G}$ on input 1^κ. Then a public and secret key pair for user 1 is generated via $(pk_1, sk_1) \xleftarrow{\$} \mathcal{K}(params)$ and a counter n for the number of users is initialized to 1. The experiment initializes an empty array $\mathbf{skp}_1$ to store the self-delegated proxy signing keys and corresponding message spaces, and empty sets $\mathbf{DU}$ and $\mathbf{CS}$. The set $\mathbf{DU}$ stores the identities of the users designated by user 1 (together with the message spaces for which they are designated). The set $\mathbf{CS}$ keeps track of the set of messages for which the adversary can produce proxy signatures by user 1 on behalf of user 1 using compromised self-delegated proxy signing keys.

Adversary $\boldsymbol{A}$ is given input pk_1, and it can make the following requests or queries, in any order and any number of times.

- (i registers pk_i) $\boldsymbol{A}$ can request to register a public key pk_i for user $i = n + 1$ by outputting pk_i. The keys are stored, counter n is incremented, and an empty array $\mathbf{skp}_i$ is created. This array will store the proxy signing keys of user 1 on behalf of user i together with the message spaces to which they correspond.
- (1 designates i) $\boldsymbol{A}$ can request to interact with user 1 running $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, for some $i \in \{2, \ldots, n\}$ and some message space ω (chosen by $\boldsymbol{A}$). In the interaction $\boldsymbol{A}$ plays the role of user i running $\mathcal{P}(pk_i, sk_i, pk_1)$. After a successful run, $\mathbf{DU}$ is set to $\mathbf{DU} \cup \{(i, \omega)\}$.
- (i designates 1) $\boldsymbol{A}$ can request to interact with user 1 running $\mathcal{P}(pk_1, sk_1, pk_i)$, for some $i \in \{2, \ldots, n\}$. In the interaction $\boldsymbol{A}$ plays the role of user i running $\mathcal{D}(pk_i, sk_i, 1, pk_1, \omega)$ for some message space ω selected by $\boldsymbol{A}$. If skp is the resulting proxy signing key, then the pair (skp, ω) is stored in the last unoccupied position of $\mathbf{skp}_i$. $\boldsymbol{A}$ does not have direct access to the elements of $\mathbf{skp}_i$.
- (1 designates 1) $\boldsymbol{A}$ can request that user 1 run the designation protocol with itself for some message space ω. $\boldsymbol{A}$ is given the transcript of the interaction. If skp is the resulting proxy signing key, the pair (skp, ω) is stored in the next available position of $\mathbf{skp}_1$.
- (exposure of the l-th proxy signing key produced during self-delegation) $\boldsymbol{A}$ can request to see $\mathbf{skp}_1[l]$ for some $l \in \mathbb{N}$. If $\mathbf{skp}_1[l]$ contains a proxy signing key and message space pair (skp, ω), then skp is returned to $\boldsymbol{A}$ and $\mathbf{CS}$ is set to $\mathbf{CS} \cup \omega$. Otherwise, $\perp$ is returned to $\boldsymbol{A}$.

- (standard signature by 1) $\boldsymbol{A}$ can query oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$ with a message M and obtain a standard signature for M by user 1, $\sigma \xleftarrow{\$} \mathcal{S}(sk_1, M)$.
- (proxy signature by 1 on behalf of i using the l-th proxy signing key) $\boldsymbol{A}$ can make a query (i, l, M), where $i \in [n]$, $l \in \mathbb{N}$ and $M \in \{0,1\}^*$, to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$. If $\mathbf{skp}_i[l]$ contains a proxy signing key and message space pair (skp, ω), we say the query is *valid* and the oracle returns $\mathcal{PS}(skp, M)$. Otherwise, we say the query is *invalid* and the oracle returns $\perp$.

Eventually, $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. The output of the experiment is determined as follows:

1. If the forgery is of the form (M, σ), where $\mathcal{V}(pk_1, M, \sigma) = 1$, and M was not queried to oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$, then return 1. [forgery of a standard signature]
2. If the forgery is of the form $(M, p\sigma, pk_i)$, for some $i \in \{2, \ldots, n\}$, where $\mathcal{PV}(pk_i, M, p\sigma) = 1$, $\mathcal{ID}(p\sigma) = 1$, and no valid query (i, l, M), for $l \in \mathbb{N}$, was made to $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, then return 1. [forgery of a proxy signature by user 1 on behalf of user $i \neq 1$]
3. If the forgery is of the form $(M, p\sigma, pk_1)$, where $\mathcal{PV}(pk_1, M, p\sigma) = 1$, $\mathcal{ID}(p\sigma) = 1$, no valid query $(1, l, M)$, for $l \in \mathbb{N}$, was made to $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, and $M \notin \mathbf{CS}$ then return 1. [forgery of a proxy signature by user 1 on behalf of user 1]
4. If the forgery is of the form $(M, p\sigma, pk_1)$, where $\mathcal{PV}(pk_1, M, p\sigma) = 1$ and for each message space ω for which $(\mathcal{ID}(p\sigma), \omega) \in \mathbf{DU}$ it holds that $M \notin \omega$ then return 1. [forgery of a proxy signature by user $i \neq 1$ on behalf of user 1; user i was not designated by user 1 to sign M]
5. Otherwise, return 0

We define the *advantage* of adversary $\boldsymbol{A}$ as

$$\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{PS},\boldsymbol{A}}(\kappa) \;=\; \Pr\left[\,\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{PS},\boldsymbol{A}}(\kappa) = 1\,\right].$$

We say that PS is a *secure proxy signature scheme* if the function $\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{PS},\boldsymbol{A}}(\cdot)$ is negligible for all adversaries $\boldsymbol{A}$ of time complexity polynomial in the security parameter κ. ▌

4 Delegation-by-Certificate Proxy Signature Schemes

Since the introduction of the proxy signature primitive [29], it has been believed that proxy signature schemes can be securely constructed from any digital signature scheme in a very simple way. Informally, a user i (with public and secret key pair (pk_i, sk_i)) can delegate its signing capability to user j (with keys pk_j, sk_j) by sending it an *warrant*. The warrant consists of the description ω of the message space for which signing is being delegated, together with a *certificate* which is signature on ω under key sk_i. Once designated, user j can create a proxy signature for a message M by computing a signature for M under its secret key sk_j and concatenating this signature with the warrant. Verification of a proxy signature involves verifying the validity of the certificate contained in the warrant relative to pk_i, verifying the validity of the signature for M relative to pk_j, and checking that M conforms to the restrictions specified in ω.

As we mentioned in the Introduction, the *raison d'être* of most of the previous work on proxy signature schemes was to improve on the efficiency of the delegation-by-certificate solution. Perhaps as a consequence, its details have never been properly pinned down. We believe that discussing this scheme in some detail is important since its conceptual simplicity and generality make it convenient for implementation in applications requiring the functionality of proxy signatures. We first note various weaknesses of a naive implementation of the scheme. Then, we propose appropriate fixes and show that the resulting scheme is indeed secure.

FLAWS AND FIXES. Consider the following chosen-message attack: The adversary requests a standard signature by user 1 for a message M. The adversary then produces a warrant certifying that user 1 can produce signatures on behalf of a user i. The signature on M together with the warrant comprise a proxy signature valid relative to the public key of user i. Thus, the adversary can forge a proxy signature by user 1 on behalf of user i. This and similar attacks can be prevented by introducing a way to differentiate between signatures created for standard signing, proxy designation, and proxy signing. For example, to sign a message M the user actually signs message $11||M$, whereas o produce a warrant by signing the message space description ω during the designation process, the user actually signs $00||\omega$. Finally, to produce a proxy signature on message M the proxy signer signs $01||M$. Verification of standard (resp., proxy) signatures is then performed by first prepending 11 (resp., 01) to the message.

This fix is insufficient. The resulting delegation-by-certificate proxy signatures still have some undesirable malleability properties. Consider the following attack: The adversary first designates user 1 as a proxy signer for a user i. Then it requests a proxy signature by 1 on behalf of i for a message M. It removes the warrant that states that user 1 can sign on behalf of i, and replaces it with one that states that 1 is allowed to sign on behalf of a user $k \neq i$. The result is a forgery of a proxy signature by user 1 on behalf of user k. Again, there is a simple fix: to sign on behalf of user i, instead of signing message $01||M$, the proxy signer signs message $01||pk_i||M$. This ties the part of the proxy signature created by the proxy signer to the designator.

Malkin et al. [28] identified a weakness of naive implementations of the delegation-by-certificate scheme that arises when the scheme is used for self-delegation, namely, an adversary that compromises the proxy signing key produced by an honest user during self-delegation can easily forge this user's standard signatures. As suggested in [28], this problem can be avoided by using a new signing/verifying key pair for each instance of self-delegation.

We are now ready to summarise the above discussion by presenting the construction of a delegation-by-certificate proxy signature scheme from any digital signature scheme.

Construction 4.1 Let $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme. The algorithms of the corresponding delegation-by-certificate proxy signature scheme $\mathsf{PS}[\mathsf{DS}] = (\mathcal{G}_1, \mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$ are defined as follows:

- The parameter- and key-generation algorithms are those of DS: $\mathcal{G}_1 = \mathcal{G}$, $\mathcal{K}_1 = \mathcal{K}$.
- A standard signature for message M is obtained by prepending 11 to the message, and signing the result using $\mathcal{S}$, i.e., $\mathcal{S}_1(sk, M) = \mathcal{S}(sk, 11||M)$.
- Verification of a signature σ for message M is done by computing $\mathcal{V}_1(pk, M, \sigma) = \mathcal{V}(pk, 11||M, \sigma)$.
- User i, in order to designate user $j \neq i$ as a proxy signer for messages in message space ω, simply sends to j the description ω of the message space, together with a *certificate* $\mathsf{cert} = \S_{sk_i}(00||j||pk_j||\omega$ (i.e. a signature on the message $00||j||pk_j||\omega$, under the secret key of user i.). The corresponding proxy signing key of user j is $skp = (sk_j, pk_i, j||pk_j||\omega, \mathsf{cert})$.

  User i, in order to designate itself as proxy signer for message space ω, runs $\mathcal{K}$ to obtain (pk'_i, sk'_i), and creates a certificate $\mathsf{cert} = \S_{sk_i}(00||i||pk'_i||\omega)$. The corresponding self-delegated proxy signing key of user i is $skp = (sk'_i, pk_i, i||pk'_i||\omega, \mathsf{cert})$.
- A proxy signature by user j on behalf of user i on message $M \in \omega$ using proxy-signing key $(sk, pk_i, j||pk||\omega, \mathsf{cert})$,[2] contains the identity j of the proxy signer, the message space description ω, the public key pk of the proxy signer, the certificate cert (a signature for $00||j||pk||\omega$ under sk_i) and a signature for $01||pk_i||M$ under sk. Formally,

  $$\mathcal{PS}((sk, pk_i, j||pk||\omega, \mathsf{cert}), M) \;=\; (j, \omega, pk, \mathsf{cert}, \mathcal{S}(sk, 01||pk_i||M)).$$

  If $M \notin \omega$ then the signing algorithm returns $\perp$.
- Proxy signature verification is defined via

  $$\mathcal{PV}(pk', M, (j, \omega, pk, \mathsf{cert}, \sigma)) \;=\; \mathcal{V}(pk', 00||j||pk||\omega, \mathsf{cert}) \wedge \mathcal{V}(pk, 01||pk'||M, \sigma) \wedge (M \in \omega).$$
- The identification algorithm is simply defined as $\mathcal{ID}((j, \omega, pk, \mathsf{cert}, \sigma)) = j$.

[2] Here $(pk, sk) = (pk_j, sk_j)$ if $i \neq j$ and $(pk, sk) = (pk'_i, sk'_i)$ if $i = j$.

The following theorem states our result about the security of proxy signature scheme $\mathsf{PS}[\mathsf{DS}]$. This result follows from a theorem stated in Section 5.2 and is proved there.

Theorem 4.2 Let DS be a secure digital signature scheme. Then the scheme $\mathsf{PS}[\mathsf{DS}]$ defined above is a secure proxy signature scheme. Concretely, let $\boldsymbol{A}$ be an adversary against $\mathsf{PS}[\mathsf{AS}]$ that makes at most q_d delegation queries, q_{sd} self-delegation queries, q_s standard signature queries, and at most q_{ps} proxy-siganture queries. Then, there exist adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ against DS such that:

$$\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{PS}[\mathsf{DS}],\boldsymbol{A}}(\kappa) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{B}}(\kappa) + \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{C}}(\kappa) + q_{sd} \cdot \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{D}}(\kappa).$$

Furthermore, adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ make at most $q_d + q_{ps} + q_s$, $q_d + q_{sd} + q_s$, q_{ps} queries to their signing oracles, respectively. Also, if the running time of $\boldsymbol{A}$ is $t_{\boldsymbol{A}}$, then those of $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are also about $t_{\boldsymbol{A}}$.

5 Aggregate-Signature-based Proxy Signature Schemes

As we mentioned in the Introduction, aggregate signatures can be used to optimize the length, and possibly the verification time of delegation-by-certificate proxy signatures. In this section we treat this matter in more detail. We first briefly review the aggregate signature primitive, its security and existing schemes (see [7] for details.) We then show how aggregate signature schemes can be used to construct proxy signature schemes.

5.1 Aggregate Signature Schemes, their Security and Constructions

Aggregate signature schemes were introduced by Boneh et al. [7], and allow construction of a single signature for a sequence of messages, out of signatures generated by distinct users for each of these messages. Formally, an aggregate signature scheme is given by a tuple of algorithms $\mathsf{AS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathcal{AV})$, where $\mathsf{DS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ is a standard digital signature scheme, called the *base* signature scheme, $\mathcal{A}$ is the *aggregation* algorithm, and $\mathcal{AV}$ is the *aggregate verification* algorithm. The aggregation algorithm takes input a sequence of public keys $pk_1, \ldots, pk_n$, messages $M_1, \ldots, M_n$ and signatures $\sigma_1, \ldots, \sigma_n$, where each signature σ_i is valid for M_i relative to public key pk_i, and outputs a *single* aggregate signature $a\sigma$. The aggregate verification algorithm takes input a sequence of public keys $pk_1, \ldots, pk_n$ and messages $M_1, \ldots, M_n$ and an aggregate signature $a\sigma$, and outputs a bit. It is required that

$$\mathcal{AV}(pk_1, \ldots, pk_n, M_1, \ldots, M_n, \mathcal{A}(pk_1, \ldots, pk_n, M_1, \ldots, M_n, \mathcal{S}(sk_1, M_1), \ldots, \mathcal{S}(sk_n, M_n))) = 1.$$

SECURITY OF AGGREGATE SIGNATURE SCHEMES. The security of aggregate signature schemes is defined via an experiment $\mathbf{Exp}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa)$ associated to scheme AS, adversary $\boldsymbol{B}$ and security parameter κ. First, system parameters $params$ are generated by running $\mathcal{G}$ on input 1^κ. Then, a public and secret key pair (pk, sk) is selected by running the key-generation algorithm $\mathcal{K}$ on input $params$, and pk is given as input to $\boldsymbol{B}$. Furthermore, $\boldsymbol{B}$ is provided with access to the signing oracle $\mathcal{S}(sk_1, \cdot)$. The goal of the adversary is to output a forgery, i.e., n public keys $pk_1, pk_2, \ldots, pk_n$, for some $n \geq 1$, a sequence of messages $M_1, \ldots, M_n$, and an aggregate signature $a\sigma$ for these messages. The experiment returns 1 (in which case we say that $\boldsymbol{B}$ wins) if $\mathcal{AV}(pk_1, \ldots, pk_n, M_1, \ldots, M_n, a\sigma) = 1$, $pk_i = pk$ for some $1 \leq i \leq n$ and $\boldsymbol{B}$ did not submit M_i to the signing oracle. The advantage of adversary $\boldsymbol{B}$ is defined by

$$\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa) = \Pr\left[\,\mathbf{Exp}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa) = 1\,\right],$$

and aggregate signature scheme AS is said to be secure if the function $\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\cdot)$ is negligible for any efficient adversary $\boldsymbol{B}$. We note that the security definition above is a slight generalization of the one of Boneh et. al. [7], which requires that $pk_1 = pk$.

CONSTRUCTIONS OF AGGREGATE SIGNATURE SCHEMES. Note that any secure standard signature scheme DS yields a secure trivial aggregate signature scheme in which the aggregation algorithm simply concatenates all signatures, and the aggregate verification algorithm simply verifies all signatures. We will refer to the aggregate signature scheme constructed this way as $\mathsf{TAS[DS]}$. A straightforward argument shows that for any efficient adversary $\boldsymbol{A}$ against $\mathsf{TAS[DS]}$ that runs in time $t_{\boldsymbol{A}}$ and makes q_S signature queries to its oracle, there exists an efficient adversary $\boldsymbol{B}$ against DS that runs in time $t_{\boldsymbol{A}}$ and that makes q_S signature queries to its oracle, such that

$$\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{TAS[DS]},\boldsymbol{A}}(\kappa) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathsf{DS},\boldsymbol{B}}(\kappa) \; ; \tag{1}$$

that is, $\mathsf{TAS[DS]}$ is secure given that the underlying signature scheme DS is secure.

We remark that sequential aggregate signature schemes [26, 24], where signing and aggregation are performed sequentially are also suitable for construction of proxy signature schemes.

5.2 Aggregate-Signature-based Proxy Signature Schemes

We sketch the construction of a proxy signature scheme from any aggregate signature scheme.

Construction 5.1 Let $\mathsf{AS} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathcal{AV})$ be an aggregate signature scheme. The algorithms of the corresponding proxy signature scheme $\mathsf{PS[AS]} = (\mathcal{G}_1, \mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$ are defined as follows.

- Algorithms $\mathcal{G}_1, \mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1, \mathcal{D}, \mathcal{P}$ use algorithms $\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}$ in the same way as those in Construction 4.1.
- If $M \in \omega$, the proxy signing algorithm $\mathcal{PS}$ uses $\mathcal{A}$ to aggregate the certificate and a proxy signature as follows:
$$\begin{aligned}&\mathcal{PS}((sk, pk_i, j||pk||\omega, \mathsf{cert}), M) \;=\\ &\qquad (j, \omega, pk, \mathcal{A}(pk_i, pk, 00||j||pk||\omega, 01||pk_i||M, \mathsf{cert}, \mathcal{S}(sk, 01||pk_i||M))).\end{aligned}$$
If $M \notin \omega$ then the signing algorithm returns $\perp$.
- The proxy verification algorithm $\mathcal{PV}$ is defined by
$$\mathcal{PV}(pk', M, (j, \omega, pk, a\sigma)) \;=\; \mathcal{AV}(pk', pk, 00||j||pk||\omega, 01||pk'||M, a\sigma) \wedge (M \in \omega).$$
- The identification algorithm is defined by $\mathcal{ID}((j, \omega, pk', a\sigma)) = j$.

The following theorem formally relates the security of the above construction to the security of the base aggregate signature scheme.

Theorem 5.2 Let AS be a secure aggregate signature scheme. Then the scheme $\mathsf{PS[AS]}$ defined above is a secure proxy signature scheme. Concretely, let $\boldsymbol{A}$ be an adversary against $\mathsf{PS[AS]}$ that makes at most q_d delegation queries, q_{sd} self-delegation queries, q_s standard signature queries, and at most q_{ps} proxy-signature queries. Then, there exist adversaries $\boldsymbol{B}$,$\boldsymbol{C}$, and $\boldsymbol{D}$ against AS, such that

$$\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{PS[AS]},\boldsymbol{A}}(\kappa) \leq \mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa) + \mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{C}}(\kappa) + q_{sd} \cdot \mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{D}}(\kappa).$$

Furthermore, adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ make at most $q_d + q_{ps} + q_s$, $q_d + q_{sd} + q_s$, q_{ps} queries to their signing oracles, respectively. Also, if the running time of $\boldsymbol{A}$ is $t_{\boldsymbol{A}}$, then the running times of $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are also about $t_{\boldsymbol{A}}$.

The proof of this theorem is in Appendix A. We now use the theorem to prove Theorem 4.2.

Proof of Theorem 4.2: Let $\mathsf{TAS[DS]}$ be the trivial aggregate signature scheme defined in Section 5.1. As we mentioned there, it is secure if DS is secure. $\mathsf{PS[TAS[DS]]}$ as defined by Construction 5.1 is exactly the delegation-by-certificate scheme $\mathsf{PS[DS]}$ as per Construction 4.1. Therefore, Theorem 5.2 implies that $\mathsf{PS[DS]}$ is secure. The concrete security reduction follows from that of Theorem 5.2 and Equation 1. ▌

In the Introduction we sketched a concrete example of using the bilinear aggregate signature scheme from [7] to build a proxy signature scheme. Since the bilinear aggregate signature scheme was proved secure in the random-oracle, assuming hardness of the Computational co-Diffie-Hellman assumption, the above theorem together with Theorem 5.2 imply that under the same assumptions, the proxy signature scheme obtained from the bilinear aggregate signature as per Construction 5.1 is a secure proxy signature scheme. The length of the corresponding proxy signature is essentially the length of the message space description plus the length of one short co-GDH signature. The use of the bilinear aggregate signature scheme also permits computational savings since the verification of a proxy signature requires 3 bilinear map computations versus 4 in the verification of a proxy signature in the co-GDH-signature-based delegation-by-certificate solution.

6 The Triple Schnorr Scheme and its Security

Kim, Park, and Won [17] proposed a proxy signature scheme based on the discrete-logarithm problem (DLP), which we call KPW. It employs Schnorr's signature scheme [35] for both standard signing and delegation, and allows the use of any signature scheme based on the hardness of the DLP for generation of proxy signatures. We make important modifications to the version of the KPW scheme in which Schnorr's signature scheme is also used for proxy signing, and prove that the resulting proxy signature scheme is secure in the random-oracle model, under the assumption of hardness of the DLP. We call this provably-secure scheme Triple Schnorr. We remark that our modifications do not affect the length of the signatures produced nor do they have a significant impact on performance. Our proof is in the basic model which does not require proofs of knowledge of secret keys during public-key registration[3].

We begin by recalling Schnorr's digital signature scheme.

6.1 Schnorr Signature Scheme and the Discrete Logarithm Assumption

A randomized polynomial-time algorithm $\mathcal{G}_{\rm dl}$ is said to be a *discrete-logarithm parameter generator* if given input 1^κ, it returns a triple (p, q, g) where p, q are primes such that $2^{\kappa-1} \le p < 2^\kappa$ (p is κ bits long) and q divides $p-1$, and $g \in \mathbb{Z}_p^*$ is an element of order q.

On input 1^κ, the Schnorr signature scheme's parameter-generation algorithm $\mathcal{G}$ runs a discrete-logarithm parameter generator $\mathcal{G}_{\rm dl}$ to obtain (p, q, g). It then selects a hash function $G : \{0,1\}^* \to \mathbb{Z}_q$ and outputs (p, q, g, G).

The key-generation algorithm $\mathcal{K}$, on input (p, q, g, G), selects a random $x \in \mathbb{Z}_q$, computes $X \leftarrow g^x \bmod p$, and outputs the pair $((p, q, g, G, X), (p, q, g, G, x))$ of public and secret keys. To simplify the notation, we will assume that the values p, q, g, G are available to all parties and we will not include them explicitly in the public and secret keys (i.e., the public and secret keys will simply be X and x, respectively).

To sign a message M, the signing algorithm $\mathcal{S}$ performs the following operations.

Pick a random $y \in \mathbb{Z}_q$; Compute a *commitment* $Y \leftarrow g^y \bmod p$
Compute a *challenge* $c \leftarrow G(M||Y)$; Compute $s \leftarrow y + c \cdot x \bmod q$
Output (Y, s) as the signature of M

To verify a signature $(\overline{Y}, \bar{s})$ for message M, the verification algorithm $\mathcal{V}$ performs the following operations.

Compute the challenge $c \leftarrow G(M||\overline{Y})$;
If $g^{\bar{s}} \equiv \overline{Y} \cdot X^c \pmod p$ then output 1 else output 0

THE DISCRETE-LOGARITHM ASSUMPTION. We recall the assumption of hardness of the discrete-logarithm problem. The advantage of algorithm $\boldsymbol{A}$ in solving the discrete-logarithm problem associated to discrete-logarithm parameter generator $\mathcal{G}_{\rm dl}$ is defined as

$$\mathbf{Adv}^{\rm dl}_{\mathcal{G}_{\rm dl},\boldsymbol{A}}(\kappa) = \Pr\left[\, (p, q, g) \xleftarrow{\$} \mathcal{G}_{\rm dl}(1^\kappa) \,;\, X \xleftarrow{\$} \langle g \rangle \,;\, y \xleftarrow{\$} \boldsymbol{A}(p, q, g, X) \;:\; g^y \equiv X \pmod p \,\right]$$

[3] The proof in the previous version of this paper [6] assumed that users prove knowledge of secret keys during public key registration.

$$\begin{array}{lcl}
\mathcal{D}(pk_i, sk_i, j, pk_j, \omega) & & \mathcal{P}(pk_j, sk_j, pk_i) \\
\mathsf{cert} \xleftarrow{\$} \mathcal{S}^G(sk_i, 0||pk_i||j||pk_j||\omega) & \xrightarrow{\omega, \mathsf{cert}} & \text{If } \mathcal{V}^G(pk_i, 0||pk_i||j||pk_j||\omega, \mathsf{cert}) = 0 \text{ then abort} \\
 & & \text{Parse cert as } (Y, s) \\
 & & c \leftarrow G(0||pk_i||j||pk_j||\omega||Y) \\
 & & r \leftarrow R(pk_i||j||pk_j||\omega||Y||c) \\
 & & t \leftarrow r \cdot sk_j + s \bmod q \\
 & & skp \leftarrow (pk_i||j||pk_j||\omega, Y, t)
\end{array}$$

Figure 1: The Triple Schnorr designation protocol run by user i as designator, and user $j \neq i$ as proxy signer.

We say that the discrete-logarithm problem associated to $\mathcal{G}_{\mathrm{dl}}$ is *hard* if for every polynomial-time algorithm $\boldsymbol{A}$, the function $\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{A}}(\cdot)$ is negligible.

The Schnorr signature scheme is known to be provably-secure in the random-oracle model, assuming the DLP associated to the underlying discrete-logarithm parameter generator is hard [34]. In the sequel, $\mathsf{S} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ denotes the Schnorr scheme. We use the notation $\mathcal{S}^G$, $\mathcal{V}^G$ to emphasize that the hash function used in the scheme is G.

6.2 Triple Schnorr Proxy Signature Scheme

We now define Triple Schnorr. For ease of comparison with KPW, the latter is described in Appendix B.

Construction 6.1 The Triple Schnorr scheme is the proxy signature scheme $\mathsf{TS} = (\mathcal{G}_\mathsf{T}, \mathcal{K}_\mathsf{T}, \mathcal{S}_\mathsf{T}, \mathcal{V}_\mathsf{T}, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$ whose constituent algorithms are defined as follows.

- The parameter-generation algorithm $\mathcal{G}_\mathsf{T}$ runs the Schnorr scheme parameter-generation algorithm $\mathcal{G}$ to get (p, q, g, G). It generates hash functions $H, R : \{0,1\}^* \to \mathbb{Z}_q$, and outputs (p, q, g, G, H, R).
- On input (p, q, g, G, H, R), the key-generation algorithm $\mathcal{K}_\mathsf{T}$ runs the Schnorr scheme key-generation algorithm $\mathcal{K}$ on (p, q, g, G) to get $((p, q, g, G, X), (p, q, g, G, x))$, and then outputs $((p, q, g, G, H, R, X), (p, q, g, G, H, R, x))$. Again, we will assume that the values p, q, g, G, H, R are available to all parties, and the public and secret keys will simply be $pk = X$ and $sk = x$, respectively.
- To sign a message M, the signing algorithm first prepends "1" to the message, and then runs the Schnorr signing algorithm with hash function G, on the result, i.e., $\mathcal{S}_\mathsf{T}(sk, M) = \mathcal{S}^G(sk, 1||M)$.
- To verify a signature σ for message M, the verification algorithm first prepends "1" to the message, and then runs the Schnorr verification algorithm with hash function G, on the result, i.e., $\mathcal{V}_\mathsf{T}(pk, M, \sigma) = \mathcal{V}^G(pk, 1||M, \sigma)$.
- In order to designate user $j \neq i$ as a proxy signer for messages in message space ω, user i sends user j the description ω and a certificate cert that is a Schnorr signature with hash function G under the secret key sk_i of user i for message $0||pk_i||j||pk_j||\omega$, i.e., $\mathsf{cert} = \mathcal{S}^G(sk_i, 0||pk_i||j||pk_j||\omega) = (Y, s)$. User j verifies this signature, and if it is valid, computes a proxy signing key as $skp = (pk_i||j||pk_j||\omega, Y, t)$, where $t = r \cdot sk_j + s \bmod q$, $r = R(pk_i||j||pk_j||\omega||Y||c)$, and $c = G(0||pk_i||j||pk_j||\omega||Y)$. See Figure 1.

  In order to designate itself for signing messages in message space ω, user i runs $\mathcal{K}$ (on input (p, q, g, G)) to obtain a new key pair (pk'_i, sk'_i). It creates a certificate cert that is a Schnorr signature with hash function G for message $0||pk_i||i||pk'_i||\omega$ under sk_i, i.e., $\mathsf{cert} = \mathcal{S}^G(sk_i, 0||pk_i||i||pk'_i||\omega) = (Y, s)$. The corresponding self-delegated proxy signing key of user i is $skp = (pk_i||i||pk'_i||\omega, Y, t)$, where $t = r \cdot sk'_i + s \bmod q$, $r = R(pk_i||i||pk'_i||\omega||Y||c)$ and $c = G(0||pk_i||i||pk'_i||\omega||Y)$.

- A proxy signature for message $M \in \omega$, on behalf of user i, produced by user j with proxy signing key $(pk_i||j||pk||\omega, Y, t)$ contains the identity j of the proxy signer, its public key pk, ω, the delegation commitment Y, and a Schnorr signature with hash function H for message $0||M||pk_i||j||pk||\omega||Y||r$ under key t, where $r = R(pk_i||j||pk||\omega||Y||c)$ and $c = G(0||pk_i||j||pk||\omega||Y)$. Formally,

```
$\mathcal{PS}((pk_i||j||pk||\omega, Y, t), M)$
    $c \leftarrow G(0||pk_i||j||pk||\omega||Y)$ ; $r \leftarrow R(pk_i||j||pk||\omega||Y||c)$
    Return $(j, pk, \omega, Y, \mathcal{S}^H(t, 0||M||pk_i||j||pk||\omega||Y||r))$
```

- To verify a proxy signature $(j, pk, \omega, Y, \sigma)$ for message M with public key pk', the proxy verification algorithm first checks that $M \in \omega$. It then computes a proxy public key as $pkp = pk^r \cdot Y \cdot pk'^c \bmod p$, where $r = R(pk'||j||pk||\omega||Y||c)$ and $c = G(0||pk'||j||pk||\omega||Y)$, and runs the Schnorr verification algorithm with hash function H, on the computed key pkp, message $0||M||pk'||j||pk||\omega||Y||r$, and signature σ, i.e.,

```
$\mathcal{PV}(pk', M, (j, pk, \omega, Y, \sigma))$
    If $M \notin \omega$ then return 0
    $c \leftarrow G(0||pk'||j||pk||\omega||Y)$ ; $r \leftarrow R(pk'||j||pk||\omega||Y||c)$ ; $pkp \leftarrow pk^r \cdot Y \cdot pk'^c \bmod p$
    Return $(M \in \omega) \wedge \mathcal{V}^H(pkp, 0||M||pk'||j||pk||\omega||Y||r, \sigma)$
```

- The proxy identification algorithm is defined as $\mathcal{ID}((j, pk, \omega, Y, \sigma)) = j$.

We observe that verification of a Triple Schnorr proxy signature requires three exponentiations modulo p. Using a simultaneous multiple exponentiation algorithm such as Algorithm 14.88 in [30], the three exponentiations can be computed at a cost of about 1.25 exponentiations. This is a significant improvement over the Schnorr-based delegation-by-certificate scheme, for which verification (using simultaneous multiple exponentiation) requires roughly 2.5 exponentiations. Standard signing and proxy signing require approximately the same amount of time in the Triple Schnorr scheme as in the Schnorr-based delegation-by-certificate scheme. Proxy designation requires one additional modular multiplication to compute the proxy signing key in the Triple Schnorr scheme.

SECURITY OF TRIPLE SCHNORR. The following theorem states our result about the security of the Triple Schnorr proxy signature scheme in the random-oracle model and without any assumptions about key registration. The proof of this theorem is quite technical and is deferred to Appendix C.

Theorem 6.2 Let $\mathcal{G}_{\mathrm{dl}}$ be a discrete-logarithm parameter generator and let $\mathsf{TS} = (\mathcal{G}_{\mathsf{T}}, \mathcal{K}_{\mathsf{T}}, \mathcal{S}_{\mathsf{T}}, \mathcal{V}_{\mathsf{T}}, (\mathcal{D}, \mathcal{P}), \mathcal{PS}, \mathcal{PV}, \mathcal{ID})$ be the associated Triple Schnorr scheme defined above. If the DLP is hard for $\mathcal{G}_{\mathrm{dl}}$, then TS is a secure proxy signature scheme in the random-oracle model.

Concretely, let $\boldsymbol{A}$ be an adversary against TS that makes at most q_G queries to random oracle G, q_R queries to random oracle R, q_H queries to random oracle H, q_d requests to be designated by user 1, q_{sd} self-delegation requests, q_s standard signature queries, and q_p proxy signature queries. Then there exist adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{E}$, and $\boldsymbol{F}$ against the discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ underlying TS such that

$$\begin{aligned}
&\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{TS},\boldsymbol{A}}(\kappa) \\
&\quad\leq\ \sqrt{q_G \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{B}}(\kappa)} \;+\; \sqrt[4]{(q_R+q_H)^6 \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{C}}(\kappa)} \;+\; \sqrt[6]{(q_G+q_H)^{10} \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{D}}(\kappa)} \\
&\qquad q_{sd} \cdot \sqrt{\delta \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{E}}(\kappa)} \;+\; \sqrt[6]{(q_G+q_H)^{10} \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{F}}(\kappa)} \;+\; \sqrt[4]{\frac{3(q_R+q_H)^6}{q}} \;+ \\
&\qquad 2 \cdot \sqrt[6]{\frac{(q_G+q_H)^{10}}{q}} \;+\; 2 \cdot \sqrt[6]{\frac{5(q_G+q_H)^{10}}{q}} \;+\; q_{sd} \cdot \sqrt{\frac{\delta}{q}} \;+\; \frac{\delta \cdot q_{sd} + 3}{q} \;+ \\
&\qquad \frac{4\Big(q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H)\Big) + q_G + 10}{q}\,, \qquad (2)
\end{aligned}$$

where q is the minimum value that can be returned as the second output by $\mathcal{G}_{\text{dl}}$.

Furthermore, if $t_{\boldsymbol{A}}$ denotes the running time of $\boldsymbol{A}$, then the running times of $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{F}$ are about $2t_{\boldsymbol{A}}$, $4t_{\boldsymbol{A}}$, $6t_{\boldsymbol{A}}$, $2t_{\boldsymbol{A}}$, and $6t_{\boldsymbol{A}}$, respectively.

The concrete security bound we got is not particularly tight, and strictly speaking, does not justify the security savings the scheme provides. However, our proof is the first that shows that the KPW scheme is secure and, of course, it does not rule out, the possibility of a tighter reduction.

7 Acknowledgements

The authors are grateful to Mihir Bellare for valuable discussions. They also thank David Pointcheval for clarifications on [34] and Gregory Neven for clarifications on [3]. The second author thanks Vadim Lyubashevsky for helpful discussions and Andrés Caicedo for suggesting the use of Hölder's inequality.

References

[1] A. Bakker, M. Steen, and A. S. Tanenbaum. A law-abiding peer-to-peer network for free-software distribution. In *IEEE International Symposium on Network Computing and Applications (NCA'01)*, 2001.

[2] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS'06*, pages 390–399. ACM Press, 2006.

[3] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a generalized forking lemma. In *CCS)*. ACM, 2006.

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*. ACM, 1993.

[5] M. Blaze and M. Strauss. Atomic proxy crpytography. In *Eurocrypt*, LNCS, 1998.

[6] A. Boldyreva, A. Palacio, and B. Warinschi. Secure proxy signature schemes for delegation of signing rights. *Cryptology ePrint Archive, Report 2003/096.*, 2003.

[7] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Eurocrypt'03*, volume 2656 of *LNCS*, 2003.

[8] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Asiacrypt '01*, volume 2248 of *LNCS*, 2001.

[9] Z. Dong, S. Liu, and K. Chen. Cryptanalysis of B.Lee-S.Kim-K.Kim proxy signature. *Cryptology ePrint Archive, Report 2003/200.*, 2003.

[10] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *CCS*, 1998.

[11] H. Ghodosi and J. Pieprzyk. Repudiation of cheating and non-repudiation of Zhang's proxy signature schemes. In *LNCS*, volume 1587. Springer-Verlag, 2001.

[12] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[13] J. Herranz and G. Saez. Revisiting fully distributed proxy signature schemes. *Cryptology ePrint Archive, Report 2003/197.*, 2003.

[14] J. Herranz and G. Saez. Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures. In *Financial Cryptography 2003*, LNCS. Springer-Verlag, 2003.

[15] A. Ivan and Y. Dodis. Proxy Cryptography Revisited. *NDSS 2003*, 2003.

[16] H. Kim, J. Baek, B. Lee, and K. Kim. Secret computation with secrets for mobile agent using one-time proxy signature. In *Cryptography and Information Security 2001*, 2001.

[17] S. Kim, S. Park, and D. Won. Proxy signatures, revisited. In *ICICS'97*, volume 1334 of *LNCS*, 1997.

[18] S. Lal and A. K. Awasthi. Proxy blind signature scheme. *Cryptology ePrint Archive, Report 2003/072.*, 2003.

[19] S. Lal and A. K. Awasthi. A scheme for obtaining a warrant message from the digital proxy signatures. *Cryptology ePrint Archive, Report 2003/073.*, 2003.

[20] B. Lee, H. Kim, and K. Kim. Strong proxy signature and its applications. In *SCIS*, 2001.

[21] J. Lee, J. Cheon, and S. Kim. An analysis of proxy signatures: Is a secure channel necessary? In M. Joye, editor, *CT-RSA'03*, volume 2612 of *LNCS*, 2003.

[22] N.-Y. Lee, T. Hwang, and C.-H. Wang. On Zhang's nonrepudiable proxy signature schemes. In *ACISP'98*, 1999.

[23] J. Leiwo, C. Hanle, P. Homburg, and A. S. Tanenbaum. Disallowing unauthorized state changes of distributed shared objects. In *SEC*, pages 381–390, 2000.

[24] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004, pages 465–485, 2006.

[25] J. Lv, J. Liu, and X. Wang. Further cryptanalysis of some proxy signature schemes. *Cryptology ePrint Archive, Report 2003/111.*, 2003.

[26] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027, pages 74–90, 2004.

[27] C. N. M. Bellare and G. Neven. Unrestricted aggregate signatures. *Cryptology ePrint Archive, Report 2006/285. Available at* `http://eprint.iacr.org/`, 2006.

[28] T. Malkin, S. Obana, and M. Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In *Eurocrypt*, LNCS, 2004.

[29] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *CCS)*. ACM, 1996.

[30] A. Menezes, P. C. van Oorschot, and S. Vanstone. *Handbook of Apllied Cryptography*. CRC Press, 1997.

[31] B. C. Neuman. Proxy based authorization and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 283–291, 1993.

[32] T. Okamoto, M. Tada, and E. Okamoto. Extended proxy signatures for smart cards. In *LNCS*, volume 1729 of *LNCS*. Springer-Verlag, 1999.

[33] H.-U. Park and L.-Y. Lee. A digital nominative proxy signature scheme for mobile communications. In *ICICS 2001*, volume 2229 of *LNCS*, 2001.

[34] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[35] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[36] K. Shum and V.-K. Wei. A strong proxy signature scheme with proxy signer privacy protection. In *WET ICE '02*, 2002.

[37] H. Sun, N.-Y. Lee, and T. Hwang. Threshold proxy signatures. In *IEE Proceedings - Computers and Digital Techniques*, volume 146, pages 259–263, 1999.

[38] H. M. Sun. An efficient nonrepudiable threshold proxy signature scheme with known signers. *Computer Communications*, 22(8):717–722, 1999.

[39] H. M. Sun. On the design of time-stamped proxy signatures with traceable receivers. In *IEE Proceedings - Computers and Digital Techniques*, 2000.

[40] H.-M. Sun and B.-T. Hsieh. Remarks on two nonrepudiable proxy signature schemes. In *Ninth National Conference on Information Security*, volume 241-246, 1999.

[41] H.-M. Sun and B.-T. Hsieh. On the security of some proxy signature schemes. *Cryptology ePrint Archive, Report 2003/068.*, 2003.

[42] Z. Tan and Z. Liu. Provably secure delegation-by-certification proxy signature schemes. *Cryptology ePrint Archive, Report 2004/148. Available at* `http://eprint.iacr.org/`, 2004.

[43] V. Varadharajan, P. Allen, and S. Black. An analysis of the proxy problem in distributed systems. In *Proceedings of 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 255–275, 1991.

[44] G. Wang, F. Bao, J. Zhou, and R. H. Deng. Security analysis of some proxy signatures. *Cryptology ePrint Archive, Report 2003/196.*, 2003.

[45] H. Wang and J. Pieprzyk. Efficient one-time proxy signatures. In *Asiacrypt'03*, volume 2894 of *LNCS*, pages 507–522, 2003.

[46] C.-K. Wu and V. Varadharajan. Modified Chinese Remainder Theorem and its application to proxy signatures. In *ICPP Workshop*, 1999.

[47] S.-M. Yen, C.-P. Hung, and Y.-Y. Lee. Remarks on some proxy signature schemes. In *Workshop on Cryptology and Information Security, 2000 ICS*, 2000.

[48] F. Zhang, R. Safavi-Naini, and C.-Y. Lin. New proxy signature, proxy blind signature and proxy ring signature schemes from bilinear pairing. *Cryptology ePrint Archive, Report 2003/104.*, 2003.

[49] K. Zhang. Nonrepudiable proxy signature schemes. *Manuscript, Available at* `http://citeseer.nj.nec.com/360090.html`, 1997.

[50] K. Zhang. Threshold proxy signature schemes. In *International Information Security Workshop*, 1997.

A Proof of Theorem 5.2

We define the following events associated to experiment $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{PS[AS]},\boldsymbol{A}}(\kappa)$.

E_1 : $\boldsymbol{A}$ outputs a forgery of the form (M,σ), where $\mathcal{V}_1(pk_1,M,\sigma)=1$, and M was not queried to oracle $\mathcal{O}_{\mathcal{S}_1}(sk_1,\cdot)$.

E_2 : $\boldsymbol{A}$ outputs a forgery of the form $(M,(1,\omega,pk_i,a\sigma),pk_i)$, where $i\neq 1$, $\mathcal{PV}(pk_i,M,\sigma)=1$, and no valid query (i,l,M) was made to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]},\cdot,\cdot,\cdot)$, for $j\in\mathbb{N}$.

E_3 : $\boldsymbol{A}$ outputs a forgery of the form $(M,(j,\omega,pk',a\sigma),pk_1)$, with $j\neq 1$, and such that for all $(j,\omega)\in\mathbf{DU}$ it is the case that $M\notin\omega$.

E_4 : $\boldsymbol{A}$ outputs a forgery of the form $(M,(1,\omega,pk',a\sigma),pk_1)$ such that pk' was not generated by user 1 during one of the executions of the self delegation protocol.

E_5 : $\boldsymbol{A}$ outputs a forgery of the form $(M,(1,\omega,pk',a\sigma),pk_1)$ such that pk' was generated by user 1 during one of the executions of the self delegation protocol.

We show that for every efficient adversary $\boldsymbol{A}$ against $\mathsf{TAS[DS]}$, there exist adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ such that

$$\Pr\left[\,E_1\vee E_2\vee E_3\,\right]\leq\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa)\ ,$$
$$\Pr\left[\,E_4\,\right]\leq\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{C}}(\kappa)\ ,$$
$$\Pr\left[\,E_5\,\right]\leq q_s\cdot\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{D}}(\kappa)\ ,$$

where q_s is the number of self delegation queries made by $\boldsymbol{A}$. Since events E_1,E_2,E_3,E_4,E_5 form a partition of the event that $\boldsymbol{A}$ wins in the $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{PS[DS]},\boldsymbol{A}}(\kappa)$, it follows that

$$\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{PS[DS]},\boldsymbol{A}}(\kappa)\leq\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa)+\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{C}}(\kappa)+q_s\cdot\mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{D}}(\kappa)\ .$$

Hence, if AS is a secure aggregate signature scheme, then $\mathsf{PS[AS]}$ is a secure proxy signature scheme.

DESCRIPTION OF ADVERSARY $\boldsymbol{B}$. Let $\boldsymbol{A}$ be an efficient adversary against $\mathsf{PS[AS]}$. We construct adversary $\boldsymbol{B}$ against AS. The adversary has access to a signing oracle under some signing key sk_1, and gets as input the corresponding verification key pk_1. Adversary $\boldsymbol{B}$ simulates the execution of adversary $\boldsymbol{A}$ against an instance of the proxy signature scheme $\mathsf{PS[AS]}$, where the keys of user 1 are (sk_1, pk_1). Adversary $\boldsymbol{B}$ works as follows:

First $\boldsymbol{B}$ initializes a counter $n = 1$ that keeps track of the number of users, creates an empty array $\mathbf{skp}_1$, and initializes sets $\mathbf{DU}$ and $\mathbf{CS}$. It then runs $\boldsymbol{A}$ on input pk_1, handling $\boldsymbol{A}$'s requests and answering $\boldsymbol{A}$'s queries as follows:

- If $\boldsymbol{A}$ requests to register a new user $i = n + 1$ by outputting pk_i, then $\boldsymbol{B}$ stores this key, increments n and creates an empty array $\mathbf{skp}_i$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by the adversary who plays the role of $\mathcal{P}(pk_i, sk_i, pk_1)$ then $\boldsymbol{B}$ makes a query $00||i||pk_i||\omega$ to its signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$ and receives an answer cert. It forwards ω, cert to $\boldsymbol{A}$ and sets $\mathbf{DU}$ to $\mathbf{DU} \cup (i, \omega)$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{P}(pk_1, sk_1, pk_i)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by the adversary who plays the role of $\mathcal{D}(pk_i, sk_i, 1, pk_1)$, then $\boldsymbol{B}$ proceeds as follows. It expects to receive from $\boldsymbol{A}$ a message of the form ω, cert, $\boldsymbol{B}$ verifies that cert is a valid signature for message $00||1||pk_1||\omega$ (i.e., it checks if $\mathcal{V}(pk_i, 00||1||pk_1||\omega, \mathsf{cert}) = 1$). If so, $\boldsymbol{B}$ stores (ω, cert) in the last unoccupied position of $\mathbf{skp}_i$. (Notice that, unlike in the real experiment, adversary $\boldsymbol{B}$ cannot store in sk_1 the actual proxy signing keys of user 1 since these keys contain sk_1. However, it is sufficient to store (ω, cert) since this information, together with access to the signing oracle under sk_1, is sufficient for producing proxy signatures).
- If $\boldsymbol{A}$ requests that user 1 run the designation protocol with itself for message space ω, $\boldsymbol{B}$ runs $\mathcal{K}$ to obtain (pk_1', sk_1'), and obtains a signature cert on $00||1||pk_1'||\omega$ from the signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$. It then forwards ω, cert to $\boldsymbol{A}$ and stores $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ in the last unoccupied position of $\mathbf{skp}_1$.
- If $\boldsymbol{A}$ requests to see $\mathbf{skp}_1[i]$ for some i, then let $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ be the i'th entry in $\mathbf{skp}_1[i]$. Adversary $\boldsymbol{B}$ forwards $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ to $\boldsymbol{A}$ and sets $\mathbf{CS}$ to $\mathbf{CS} \cup \omega$.
- If $\boldsymbol{A}$ queries its oracle $\mathcal{O}_{\mathcal{S}_1}(sk_1, \cdot)$ with a message M, $\boldsymbol{B}$ makes query $11||M$ to its own signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$ and forwards the response to $\boldsymbol{A}$.
- If $\boldsymbol{A}$ makes a query (i, l, M), where $i \in [n]$, $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u \in [n]}, \cdot, \cdot, \cdot)$, $\boldsymbol{B}$ responds as follows. If $i = 1$ and $\mathbf{skp}_1[l]$ is not defined then return $\perp$ to $\boldsymbol{A}$. Otherwise, compute and return to $\boldsymbol{B}$ the quantity $\mathcal{PS}(\mathbf{skp}_1[l], M)$. If $i \neq 1$ and $\mathbf{skp}_i[l]$ is not defined, it returns $\perp$ to $\boldsymbol{A}$. Otherwise, let (ω, cert) be the content of this position. $\boldsymbol{B}$ submits to the signing oracle $(01||pk_i||M)$ and obtains in return σ. The proxy signature that $\boldsymbol{B}$ returns to $\boldsymbol{A}$ is $(1, \omega, pk_1, a\sigma)$ where $a\sigma = \mathcal{A}(pk_i, pk_1, 00||1||pk_1||\omega, 01||pk_i||M, \mathsf{cert}, \sigma)$.

Eventually, $\boldsymbol{A}$ outputs an attempted forgery. Adversary $\boldsymbol{B}$ computes its attempted forgery as follows:

1. If $\boldsymbol{A}$ outputs a forgery of the form (M, σ) then the forgery output by $\boldsymbol{B}$ is $(11||M, \sigma)$.
2. If $\boldsymbol{A}$ outputs $(M, (1, \omega, pk_1, a\sigma), pk_i)$ and $i \neq 1$ then adversary $\boldsymbol{B}$ outputs $(pk_i, pk_1, 00||1||pk_1||\omega, 01||pk_i||M, a\sigma)$.
3. If $\boldsymbol{A}$ outputs a forgery $(M, (j, \omega, pk_j, \sigma), pk_1)$ such that $\mathcal{ID}((j, \omega, pk_j, \sigma)) = j$ and j is such that for all ω with $(j, \omega) \in \mathbf{DU}$ $M \notin \omega$ then $\boldsymbol{B}$ outputs $(pk_j, 00||1||pk_j||\omega, 01||pk_1||M, \sigma)$.

ANALYSIS OF ADVERSARY $\boldsymbol{B}$. It is clear that the view of $\boldsymbol{A}$ in the simulated experiment is identical to that in the experiment $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{PS[AS]},\boldsymbol{A}}(\kappa)$. We next argue that if either of events E_1, E_2 or E_3 occurs during the execution of $\boldsymbol{A}$ then adversary $\boldsymbol{B}$ wins in the experiment $\mathbf{Exp}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa)$.

1. If event E_1 occurs, then M, σ are such that $\mathcal{V}_1(pk_1, 11||M, \sigma) = 1$, and the forgery attempted by $\boldsymbol{B}$ is $(11||M, \sigma)$. We only need to argue that $11||M$ was not in a query of $\boldsymbol{B}$ to its signing oracle. Since the forgery output by $\boldsymbol{A}$ is valid for the $\mathbf{Exp}^{\text{ps-uf}}_{,\mathsf{PS[AS]}}\boldsymbol{A}(\kappa)$ experiment, it follows that $\boldsymbol{A}$ (in that experiment) never queried M to its standard signing oracle. It is immediate that $\boldsymbol{B}$ did not need to query $11||M$ to its own signing oracle, and therefore the forgery outputed by $\boldsymbol{B}$ is valid.

2. If event E_2 occurs, then the forgery output by $\boldsymbol{A}$ is of the form $(M, (1, \omega, pk_i, a\sigma), pk_i)$ for some $i \neq 1$, with $a\sigma$ a valid aggregate signature on messages $00||1||pk_1||\omega$ and $01||pk_i||M$ with respect to keys pk_i and pk_1, i.e. $(\mathcal{AV}(pk_i, pk_1, 00||1||pk_1||\omega, 01||pk_i||M, a\sigma)) = 1$, so the forgery attempted by $\boldsymbol{B}$ satisfies the verification requirement. We only need to argue that $\boldsymbol{B}$ did not query the message $01||pk_i||M$ to its signing oracle. From the description of adversary $\boldsymbol{B}$ it is clear that the only circumstance when $\boldsymbol{B}$ issues such a query is when $\boldsymbol{A}$ makes a query of the form (i, l, M) for some $l \in \mathbb{N}$ to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, which is not the case, as such a query would invalidate the forgery output by $\boldsymbol{A}$.

3. If event E_3 occurs and the forgery output by $\boldsymbol{A}$ of the form $M, (1, \omega, pk_j, a\sigma), pk_1)$, then $a\sigma$ is a valid aggregate signature on message $00||1||pk_j||\omega$ and $01||pk_1||M$ relative to pk_1 and pk_j respectively, so $\mathcal{AV}(pk_1, pk_j, 00||1||pk_j||\omega, 01||pk_1||M) = 1$. The forgery output by $\boldsymbol{B}$ satisfies thus the verification equation. It remains to argue that this forgery i.e. $(pk_1, pk_j, 00||j||pk_j||\omega, 01||pk_1||M, \sigma)$ is valid, in the sense that $\boldsymbol{B}$ did not query message $00||j||pk||\omega$ to its oracle. Notice from the description of $\boldsymbol{B}$ that this query only needs to be issued when $\boldsymbol{A}$ requests that user 1 designates user j as a proxy signer on message space ω. Such a query does not occur, since for any ω for which $(j, \omega) \in \mathbf{DU}$ it holds that $M \notin \omega$.

Putting the above together, we have that

$$\Pr\left[\, E_1 \vee E_2 \vee E_3 \,\right] \leq \mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{B}}(\kappa)\ .$$

Also, notice that adversary $\boldsymbol{B}$ needs to query its oracle onlye to answer the self-delegation, proxy signature, and standard signature queries of $\boldsymbol{A}$, that is at most $q_d + q_{ps} + q_s$ times.

DESCRIPTION OF ADVERSARY $\boldsymbol{C}$. First $\boldsymbol{C}$ initializes a counter $n = 1$ that keeps track of the number of users, creates an empty array $\mathbf{skp}_1$, and initializes sets $\mathbf{DU}$ and $\mathbf{CS}$. It then runs $\boldsymbol{A}$ on input pk_1 (the verification key that correspond to the signing key used by $\boldsymbol{C}$'s signing oracle), handling $\boldsymbol{A}$'s requests and answering $\boldsymbol{A}$'s queries as follows:

- If $\boldsymbol{A}$ requests to register a new user $i = n + 1$ by outputting pk_i, then $\boldsymbol{C}$ stores this key, increments n and creates an empty array $\mathbf{skp}_i$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary messages space chosen by the adversary who plays the role of $\mathcal{P}(pk_i, sk_i, pk_1)$ then $\boldsymbol{C}$ makes a query $00||i||pk_i||\omega$ to its signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$ and receives an answer cert. It forwards ω, cert to $\boldsymbol{A}$ and sets $\mathbf{DU}$ to $\mathbf{DU} \cup (i, \omega)$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{P}(pk_1, sk_1, pk_i)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by the adversary who plays the role of $\mathcal{D}(pk_i, sk_i, 1, pk_1, \omega)$, then $\boldsymbol{C}$ proceeds as follows. It expects to receive from $\boldsymbol{A}$ a message of the form ω, cert, $\boldsymbol{C}$ verifies that cert is a valid signature for message $00||1||pk_1||\omega$ (i.e., it checks if $\mathcal{V}(pk_i, 00||1||pk_1||\omega, \mathsf{cert}) = 1$). If so, $\boldsymbol{C}$ stores (ω, cert) in the last unoccupied position of $\mathbf{skp}_i$. (Notice that $\boldsymbol{C}$ instead of storing the proxy signing keys of user 1 for other users, $\boldsymbol{C}$ only stores the pairs (ω, cert) in $\mathbf{skp}_i$ since these proxy signing keys contain sk_1 which $\boldsymbol{C}$ does not have. However, the information stored in $\mathbf{skp}_i$ is sufficient to create proxy signatures using the signing oracle of $\boldsymbol{C}$).

- If $\boldsymbol{A}$ requests that user 1 run the designation protocol with itself for message space ω, $\boldsymbol{C}$ runs $\mathcal{K}$ to obtain (pk_1', sk_1') obtains signature cert on $00||1||pk_1'||\omega$ from the signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$. It then forwards ω, cert to $\boldsymbol{A}$ and stores $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ in the last unoccupied position of $\mathbf{skp}_1$.
- If $\boldsymbol{A}$ requests to see $\mathbf{skp}_1[i]$ for some i , then let $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ be the i'th entry in $\mathbf{skp}_1[i]$. Adversary $\boldsymbol{C}$ forwards $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ to $\boldsymbol{A}$ and sets $\mathbf{CS}$ to $\mathbf{CS} \cup \omega$.
- If $\boldsymbol{A}$ queries its oracle $\mathcal{O}_{\mathcal{S}_1}(sk_1, \cdot)$ with a message M, $\boldsymbol{C}$ makes query $11||M$ to its own signing oracle $\mathcal{O}_{\mathcal{S}}(sk_1, \cdot)$ and forwards the response to $\boldsymbol{A}$.
- If $\boldsymbol{A}$ makes a query (i, l, M), where $i \in [n]$, $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, $\boldsymbol{C}$ responds as follows. If $i = 1$ and $\mathbf{skp}_1[l]$ is not defined then return $\perp$ to $\boldsymbol{A}$. Otherwise, compute and return to $\boldsymbol{C}$ the quantity $\mathcal{PS}(\mathbf{skp}_1[l], M)$. If $i \neq 1$ and $\mathbf{skp}_i[l]$ is not defined, it returns $\perp$ to $\boldsymbol{A}$. Otherwise, let (ω, cert) be the content of this position in $\mathbf{skp}_i[l]$. Adversary $\boldsymbol{C}$ submits to the signing oracle $(01||pk_i||M)$ and obtains in return σ. The proxy signature that $\boldsymbol{C}$ returns to $\boldsymbol{A}$ is $(1, \omega, pk_1, a\sigma)$ to $\boldsymbol{A}$ where $a\sigma = \mathcal{A}(pk_i, pk_1, 00||1||pk_1||\omega, 01||pk_i||M, \mathsf{cert}, \sigma)$.

Eventually, $\boldsymbol{A}$ outputs an attempted forgery. If the forgery that is output by $\boldsymbol{A}$ is of the form $(M, (1, \omega, pk', a\sigma), pk_1)$ with $a\sigma$ an aggregate signature on message $00||1||pk'||\omega$ and $01||pk_1||M$ under keys pk_1, and pk', and pk' had not been used by user 1 in the self-delegation protocol, then the attepmpted forery of $\boldsymbol{C}$ is $(pk_1, pk', 00||1||pk'||\omega, 01||pk_1||M, a\sigma)$. If the forgery attempted by $\boldsymbol{A}$ does not have this form, then adversary $\boldsymbol{C}$ aborts.

ANALYSIS OF ADVERSARY $\boldsymbol{C}$. It is easy to see that the simulation of the experiment for security of proxy signatures that $\boldsymbol{C}$ carries out for $\boldsymbol{A}$ is perfect. Assume that event E_4 occurs in this simulation. Then adversary $\boldsymbol{C}$ does not abort, and outputs $(pk_1, pk', 00||1||pk'||\omega, 01||pk_1||M, a\sigma)$, as its attempted forgery. Since the forgery output by $\boldsymbol{A}$, $(M, (1, \omega, pk', a\sigma), pk_1)$ is valid, then $a\sigma$ is a valid aggregate signature on messages $00||1||pk'||\omega, 01||pk_1||M$ with respect to keys pk_1, pk'. It remains to argue that the forgery is valid in the sense that $00||1||pk'||\omega$ has not been queried to the signing oracle of $\boldsymbol{C}$. This is clearly true because such a query occurs only if user 1 uses pk' for self delegation which, by the definition of event E_4, did not happened.

Finally, notice that adversary $\boldsymbol{C}$ needs to query its oracle onlye to answer the delegation, self-delegation, and standard signature queries of $\boldsymbol{A}$, that is at most $q_d + q_{ps} + q_s$ times.

DESCRIPTION OF ADVERSARY $\boldsymbol{D}$. Adversary $\boldsymbol{D}$ is against the aggregate signature scheme AS, and as such it has access to a signing oracle under sk and has as input the corresponding verification key pk. Adversary $\boldsymbol{D}$ runs internally adversary $\boldsymbol{A}$ for which it simulates its environment. The simulation is as follows. First, $\boldsymbol{D}$ selects a random index $t \in \{1, 2, \ldots, q_s\}$ (virtually selecting one of the self-delegation requests of $\boldsymbol{A}$). Then, $\boldsymbol{D}$ initializes a counter $n = 1$ that keeps track of the number of users, creates an empty array $\mathbf{skp}_1$, and initializes sets $\mathbf{DU}$ and $\mathbf{CS}$. Next, it generates signing/verication keys (sk_1, pk_1) for user 1. Then, $\boldsymbol{D}$ executes adversary $\boldsymbol{A}$ on input pk_1, and intercepts and its requests which $\boldsymbol{D}$ handles as follows.

- If $\boldsymbol{A}$ requests to register a new user $i = n + 1$ by outputting pk_i, then $\boldsymbol{D}$ stores this key, increments n and creates an empty array $\mathbf{skp}_i$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by the adversary who plays the role of $\mathcal{P}(pk_i, sk_i, pk_1, \omega)$ then $\boldsymbol{D}$ produces a signature cert on $00||i||pk_i||\omega$ under key sk_1 and it sends (ω, cert) to $\boldsymbol{A}$. Then, it sets $\mathbf{DU}$ to $\mathbf{DU} \cup (i, \omega)$.
- If $\boldsymbol{A}$ requests to interact with $\mathcal{P}(pk_1, sk_1, pk_i)$, where $i \in \{2, \ldots, n\}$ and ω is an arbitrary message space chosen by the adversary who plays the role of $\mathcal{D}(pk_i, sk_i, 1, pk_1, \omega)$, then $\boldsymbol{D}$ proceeds as follows. It expects to receive from $\boldsymbol{A}$ a message of the form ω, cert, $\boldsymbol{D}$ verifies that cert is a valid signature for message $00||1||pk_1||\omega$ (i.e., it checks if $\mathcal{V}(pk_i, 00||1||pk_1||\omega, \mathsf{cert}) = 1$). If so, $\boldsymbol{D}$ stores $(sk_1, pk_1, 1||pk_i||\omega, \mathsf{cert})$ in the last unoccupied position of $\mathbf{skp}_i$.
- When $\boldsymbol{A}$ asks that user 1 runs the self-delegation protocol, $\boldsymbol{D}$ proceeds as follows. For all but for the t'th self-delegation requests that $\boldsymbol{A}$ makes, adversary $\boldsymbol{D}$ runs $\mathcal{K}$ to obtain (pk_1', sk_1'), and if ω is the mes-

sage space for which $\boldsymbol{A}$ requests self-delegation, it obtains a signature cert on the message $00||1||pk_1'||\omega$ under the signing key sk_1. It then forwards ω, cert to $\boldsymbol{A}$ and stores $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ in the last unoccupied position of $\mathbf{skp}_1$.

For the t'th self-designation query, $\boldsymbol{D}$ produces a signature cert on the message $00||1||pk||\omega$ using the signing key sk_1. Here pk is the verification key that $\boldsymbol{D}$ takes as input. Then, $\boldsymbol{D}$ forwards (ω, cert) to $\boldsymbol{A}$.

- If $\boldsymbol{A}$ requests to see $\mathbf{skp}_1[i]$ for some $i \neq t$, then let $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ be the i'th entry in $\mathbf{skp}_1[i]$. Adversary $\boldsymbol{D}$ forwards $(sk_1', pk_1, 1||pk_1'||\omega, \mathsf{cert})$ to $\boldsymbol{A}$ and sets $\mathbf{CS}$ to $\mathbf{CS} \cup \omega$. If $\boldsymbol{A}$ requests $\mathbf{skp}_1[t]$, then $\boldsymbol{D}$ aborts its execution.
- If $\boldsymbol{A}$ queries its oracle $\mathcal{O}_{\mathcal{S}_1}(sk_1, \cdot)$ with a message M, $\boldsymbol{D}$ produces a signature on $11||M$ using sk_1 and forwards it to $\boldsymbol{A}$.
- If $\boldsymbol{A}$ makes a query (i, l, M), where $i \in [n]$, $l \in \mathbb{N}$, $M \in \{0,1\}^*$, to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, $\boldsymbol{D}$ responds as follows.
  1. If $i = 1$ and $l \neq t$ or $i \neq 1$ and $\mathbf{skp}_1[l]$ is not defined then return $\perp$ to $\boldsymbol{A}$.
  2. if $i = 1$ and $l \neq t$, or $i \neq 1$ and $\mathbf{skp}_1[l]$ is defined, then return to $\boldsymbol{A}$ the quantity $\mathcal{PS}(\mathbf{skp}_1[l], M)$.
  3. if $i = 1$, $l = t$, then $\boldsymbol{D}$ proceeds as follows: it sends a query $(01||pk_1||M)$ to its signing oracle, and obtains in return a signature σ under the key sk. The proxy signature that $\boldsymbol{D}$ returns to $\boldsymbol{A}$ is $(1, \omega, pk_1, a\sigma)$ where $a\sigma = \mathcal{A}(pk_1, pk, 00||1||pk||\omega, 01||pk_1||M, \mathsf{cert}, \sigma)$.

Eventually, $\boldsymbol{A}$ outputs an attempted forgery $(M, (1, \omega, pk, a\sigma), pk_1)$. If pk is not the key used in the t'th self-delegation request of $\boldsymbol{A}$, then $\boldsymbol{D}$ aborts. Otherwise, the forgery output by $\boldsymbol{D}$ is $(pk_1, pk, 00||1||pk||\omega, 01||pk_1||M, a\sigma)$.

ANALYSIS OF ADVERSARY $\boldsymbol{D}$. The simulation that $\boldsymbol{D}$ caries for $\boldsymbol{A}$ is perfect, provided that $\boldsymbol{A}$ does not request to see the t'th proxy key resulted from self-delegation. This is certainly the case whenever event E_5 occurs (i.e. the forgery output by $\boldsymbol{A}$ is of the form $(M, (1, \omega, pk, a\sigma), pk_1)$, where pk was generated by user 1 during one of the self delegation requests), and the index t selected at random by $\boldsymbol{D}$ is precisely the index of the self-delegation request where pk was generated (as otherwise the forgery output by $\boldsymbol{A}$ would be invalid). Since the forgery output by $\boldsymbol{A}$ is valid, then $a\sigma$ is a valid aggregate signature on messages $00||1||pk||\omega, 01||pk_1||M$ with respect to keys pk_1 and pk. It remains to argue that the forgery output by $\boldsymbol{D}$ is valid in the sense that $01||pk_1||M$ had not been queried by $\boldsymbol{D}$ to its signing oracle. This query is only made if $\boldsymbol{A}$ sends $(1, t, M)$ to its oracle $\mathcal{O}$. However, $\boldsymbol{A}$ does not make this query since otherwise its forgery would be invalid. Therefore, whenever event E_5 occurs and adversary $\boldsymbol{D}$ guesses correctly the index t of the self delegation request where pk is generated, then adversary $\boldsymbol{D}$ wins in $\mathbf{Exp}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{D}}(\kappa)$, i.e.:

$$\frac{1}{q_s} \cdot \Pr[\, E_5 \,] \leq \mathbf{Adv}^{\text{ag-uf}}_{\mathsf{AS},\boldsymbol{D}}(\kappa) \; .$$

It is immediate that adversary $\boldsymbol{B}$ needs to query its oracle onlye to answer the proxy signature queries of $\boldsymbol{A}$, that is at most q_{ps} times.

Finally, notice that if the running time of $\boldsymbol{A}$ is $t_{\boldsymbol{A}}$, then the running times of $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ are also about $t_{\boldsymbol{A}}$.

B KPW Proxy Signature Scheme

The variant of KPW in which Schnorr's signature scheme is used for standard signing, delegation, and proxy signing is the proxy signature scheme $\mathsf{KPW} = (\mathcal{G}, \mathcal{K}, \mathcal{S}, \mathcal{V}, (\mathcal{D}_{\text{KPW}}, \mathcal{P}_{\text{KPW}}), \mathcal{PS}_{\text{KPW}}, \mathcal{PV}_{\text{KPW}}, \mathcal{ID}_{\text{KPW}})$, where $\mathcal{G}$, $\mathcal{K}$, $\mathcal{S}$, and $\mathcal{V}$ are the algorithms of Schnorr's digital signature scheme (see Section 6.1) and the remaining algorithms are defined as follows.

$\mathcal{D}_{\text{KPW}}(pk_i, sk_i, j)$		$\mathcal{P}_{\text{KPW}}(pk_j, sk_j, pk_i)$
$\mathsf{cert} \xleftarrow{\$} \mathcal{S}^G(sk_i, \omega)$		
	$\xrightarrow{\omega, \mathsf{cert}}$	
		If $\mathcal{V}^G(pk_i, \omega, \mathsf{cert}) = 0$ then abort Parse cert as (Y, s) $t \leftarrow G(Y\|\|\omega) \cdot sk_j + s \bmod q$ $skp \leftarrow (j, pk_j, \omega, Y, t)$

$\mathcal{PS}_{\text{KPW}}((j, pk_j, \omega, Y, t), M)$	$\mathcal{PV}_{\text{KPW}}(pk, M, (\sigma, \omega, Y, j, pk_j))$	$\mathcal{ID}_{\text{KPW}}((\sigma, \omega, Y, j, pk_j))$
$\sigma \leftarrow \mathcal{S}^G(t, M)$ Return $(\sigma, \omega, Y, j, pk_j)$	$pkp \leftarrow (pk \cdot pk_j)^{G(Y\|\|\omega)} \cdot Y \bmod p$ Return $\mathcal{V}^G(pkp, M, \sigma)$	Return j

C Proof of Theorem 6.2

We begin by recalling the General Forking Lemma of Bellare and Neven [2] which we will use in the proof of Theorem 6.2.

Lemma C.1 [General Forking Lemma [2]] Fix $\alpha \in \mathbb{Z}^+$ and a set S such that $|S| \geq 2$. Let $\boldsymbol{Y}$ be a randomized algorithm that on input a string x and elements $s_1, \ldots, s_\alpha \in S$, returns a pair (I, σ) consisting of an integer $0 \leq I \leq \alpha$ and a string σ. The *forking algorithm* $\boldsymbol{F_Y}$ associated to $\boldsymbol{Y}$ is defined as follows:

```
Algorithm $\boldsymbol{F_Y}(x)$
    Pick coins $\rho$ for $\boldsymbol{Y}$ at random
    $s_1, \ldots, s_\alpha \xleftarrow{\$} S$ ; $(I, \sigma) \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_\alpha; \rho)$
    If ( $I = 0$ ) then return $(0, \varepsilon, \varepsilon)$
    $s'_I, \ldots, s'_\alpha \xleftarrow{\$} S$ ; $(I', \sigma') \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_{I-1}, s'_I, \ldots, s'_\alpha; \rho)$
    If ( $I' = I$ and $s'_I \neq s_I$ ) then return $(1, \sigma, \sigma')$ else return $(0, \varepsilon, \varepsilon)$
```

Let IG be a randomized algorithm that takes no input and returns a string. Let

$$\begin{aligned} acc &= \Pr\left[x \xleftarrow{\$} \mathsf{IG} \,;\, s_1, \ldots, s_\alpha \xleftarrow{\$} S \,;\, (I, \sigma) \xleftarrow{\$} \mathbf{Y}(x, s_1, \ldots, s_\alpha) \;:\; I \geq 1 \right] \\ frk &= \Pr\left[x \xleftarrow{\$} \mathsf{IG} \,;\, (b, \sigma, \sigma') \xleftarrow{\$} \boldsymbol{F_Y}(x) \;:\; b = 1 \right] . \end{aligned}$$

Then

$$frk \;\geq\; acc \cdot \left(\frac{acc}{\alpha} - \frac{1}{|S|} \right) .$$

Alternatively,

$$acc \;\leq\; \sqrt{\alpha \cdot frk} \;+\; \frac{\alpha}{|S|} \,.$$

We also use a related lemma which considers an algorithm $\boldsymbol{Y}$ that outputs a pair of integers, rather than a single integer, to indicate possible forking positions. The lemma involves a forking algorithm that rewinds $\boldsymbol{Y}$ multiple times. The proof of this lemma uses Jensen's inequality and a corollary of Hölder's inequality. We first recall these and then state our forking lemma.

Lemma C.2 [Jensen's inequality] If f is a convex[4] function and X is a random variable then

$$\mathbf{E}[f(X)] \quad \geq \quad f(\mathbf{E}[X]) \; .$$

Lemma C.3 [Hölder's inequality] Let $n \geq 1$ be an integer, $1 \leq p, q < \infty$ with $1/p + 1/q = 1$ and $x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{R}$. Then

$$\sum_{k=1}^{n} |x_k y_k| \quad \leq \quad \left(\sum_{k=1}^{n} |x_k|^p \right)^{1/p} \left(\sum_{k=1}^{n} |y_k|^q \right)^{1/q} \; .$$

Corollary C.4 Let $n \geq 1$ be an integer, $1 \leq p, q < \infty$ with $1/p + 1/q = 1$, and $x_1, \ldots, x_n \geq 0$ real numbers. Then

$$\sum_{k=1}^{n} x_k^p \quad \geq \quad \frac{1}{n^{p/q}} \left(\sum_{k=1}^{n} x_k \right)^{p} \; .$$

Proof: Follows from Lemma C.3 with $y_1 = \cdots = y_n = 1$, by raising both sides of the inequality to the power p and rearranging terms. ▌

The following result may be of independent interest.

Lemma C.5 [Multiple-Forking Lemma] Fix $\alpha \in \mathbb{Z}^+$ and a set S such that $|S| \geq 2$. Let $\boldsymbol{Y}$ be a randomized algorithm that on input a string x and elements $s_1, \ldots, s_\alpha \in S$, returns a triple (I, J, σ) consisting of two integers $0 \leq J < I \leq \alpha$ and a string σ. Let $n \geq 1$ be an odd integer. The *multiple-forking algorithm* $\boldsymbol{MF}_{\boldsymbol{Y},n}$ associated to $\boldsymbol{Y}$ and n is defined as follows, where x is a string:

```
Algorithm $\boldsymbol{MF}_{\boldsymbol{Y},n}(x)$
    Initialize an empty array results$[0 \ldots n]$
    Pick coins $\rho$ for $\boldsymbol{Y}$ at random
    $s_1, \ldots, s_\alpha \stackrel{\$}{\leftarrow} S$ ; $(I, J, \sigma_0) \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_\alpha; \rho)$
    If ( $I = 0$ or $J = 0$ ) then return $(0, \text{results})$
    $s^1_I, \ldots, s^1_\alpha \stackrel{\$}{\leftarrow} S$ ; $(I_1, J_1, \sigma_1) \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_{I-1}, s^1_I, \ldots, s^1_\alpha; \rho)$
    If ( $(I_1, J_1) \neq (I, J)$ or $s^1_I = s_I$ ) then return $(0, \text{results})$
    $i \leftarrow 2$
    While ( $i < n$ ) do
        $s^i_J, \ldots, s^i_\alpha \stackrel{\$}{\leftarrow} S$ ; $(I_i, J_i, \sigma_i) \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_{J-1}, s^i_J, \ldots, s^i_\alpha; \rho)$
        If ( $(I_i, J_i) \neq (I, J)$ or $s^i_J = s^{i-1}_J$ ) then return $(0, \text{results})$
        $s^{i+1}_I, \ldots, s^{i+1}_\alpha \stackrel{\$}{\leftarrow} S$ ; $(I_{i+1}, J_{i+1}, \sigma_{i+1}) \leftarrow \boldsymbol{Y}(x, s_1, \ldots, s_{J-1}, s^i_J, \ldots, s^i_{I-1}, s^{i+1}_I, \ldots, s^{i+1}_\alpha; \rho)$
        If ( $(I_{i+1}, J_{i+1}) \neq (I, J)$ or $s^{i+1}_I = s^i_I$ ) then return $(0, \text{results})$
        $i \leftarrow i + 2$
    EndWhile
    For $i = 0$ to $n$ do
        results$[i] \leftarrow \sigma_i$
    EndFor
    Return $(1, \text{results})$
```

[4] A function f is convex on an interval $[a, b]$ if for any two points x_1 and x_2 in $[a, b]$ and any λ, where $0 \leq \lambda \leq 1$, $f(\lambda x_1 + (1 - \lambda) x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)$. If f has a second derivative in $[a, b]$, then a necessary and sufficient condition for it to be convex on that interval is that the second derivative $f''(x) \geq 0$ for all x in $[a, b]$.

Let IG be a randomized algorithm that takes no input and returns a string. Let

$$\begin{aligned} acc &= \Pr\left[\, x \stackrel{\$}{\leftarrow} \mathsf{IG}\, ;\ s_1,\ldots,s_\alpha \stackrel{\$}{\leftarrow} S\, ;\ (I,J,\sigma) \stackrel{\$}{\leftarrow} \boldsymbol{Y}(x,s_1,\ldots,s_\alpha)\ :\ I \ge 1 \wedge J \ge 1 \,\right] \\ frk &= \Pr\left[\, x \stackrel{\$}{\leftarrow} \mathsf{IG}\, ;\ (b,\mathsf{results}) \stackrel{\$}{\leftarrow} \boldsymbol{MF}_{\boldsymbol{Y},n}(x)\ :\ b = 1 \,\right]. \end{aligned}$$

Then

$$frk \ \ge\ acc \cdot \left(\frac{acc^n}{\alpha^{2n}} - \frac{n}{|S|} \right). \tag{3}$$

Consequently,

$$acc \ \le\ \sqrt[n+1]{\alpha^{2n}\cdot frk} \ +\ \sqrt[n+1]{\frac{n\cdot\alpha^{2n}}{|S|}}\ . \tag{4}$$

Proof: Fix a string x. Let

$$\begin{aligned} acc(x) &= \Pr\left[\, s_1,\ldots,s_\alpha \stackrel{\$}{\leftarrow} S\, ;\ (I,J,\sigma) \stackrel{\$}{\leftarrow} \boldsymbol{Y}(x,s_1,\ldots,s_\alpha)\ :\ I \ge 1 \wedge J \ge 1 \,\right] \\ frk(x) &= \Pr\left[\, (b,\mathsf{results}) \stackrel{\$}{\leftarrow} \boldsymbol{MF}_{\boldsymbol{Y},n}(x)\ :\ b = 1 \,\right]. \end{aligned}$$

Then, with probabilities taken over the randomness of $\boldsymbol{MF}_{\boldsymbol{Y},n}$, we have

$$\begin{aligned} & frk(x) \\ &\quad = \Pr\left[\, (I_n,J_n) = (I_{n-1},J_{n-1}) = \cdots = (I_1,J_1) = (I,J) \wedge I \ge 1 \wedge J \ge 1 \wedge \right. \\ &\qquad\qquad \left. s_I^n \ne s_I^{n-1} \wedge s_J^{n-1} \ne s_J^{n-2} \wedge \cdots \wedge s_J^2 \ne s_J^1 \wedge s_I^1 \ne s_I \right] \\ &\quad \ge \Pr\left[\, (I_n,J_n) = (I_{n-1},J_{n-1}) = \cdots = (I_1,J_1) = (I,J) \wedge I \ge 1 \wedge J \ge 1 \,\right] \\ &\qquad - \Pr\left[\, I \ge 1 \wedge J \ge 1 \wedge (\, s_I^n = s_I^{n-1} \vee s_J^{n-1} = s_J^{n-2} \vee \cdots \vee s_J^2 = s_J^1 \vee s_I^1 = s_I \,) \,\right]. \end{aligned} \tag{5}$$

We compute the second term as follows.

$$\begin{aligned} & \Pr\left[\, I \ge 1 \wedge J \ge 1 \wedge (\, s_I^n = s_I^{n-1} \vee s_J^{n-1} = s_J^{n-2} \vee \cdots \vee s_J^2 = s_J^1 \vee s_I^1 = s_I \,) \,\right] \\ &\quad = \Pr\left[\, I \ge 1 \wedge J \ge 1 \,\right] \cdot \Pr\left[\, s_I^n = s_I^{n-1} \vee s_J^{n-1} = s_J^{n-2} \vee \cdots \vee s_J^2 = s_J^1 \vee s_I^1 = s_I \,\right] \\ &\quad = \frac{n\cdot\Pr\left[\, I \ge 1 \wedge J \ge 1 \,\right]}{|S|} \\ &\quad = \frac{n\cdot acc(x)}{|S|}\ . \end{aligned} \tag{6}$$

We will now show that

$$\Pr\left[\, (I_n,J_n) = (I_{n-1},J_{n-1}) = \cdots = (I_1,J_1) = (I,J) \wedge I \ge 1 \wedge J \ge 1 \,\right] \ \ge\ acc(x)^{n+1}/\alpha^{2n}\,.$$

For convenience, we will use the following shorthand:

symbol	represents	symbol	represents	symbol	represents
T	$\rho, s_1, \ldots, s_{J-1}$	U_0	$s_J, \ldots, s_{I-1}$	V_0	$s_I, \ldots, s_\alpha$
		U_1	$s^2_J, \ldots, s^2_{I-1}$	V_1	$s^1_I, \ldots, s^1_\alpha$
		U_2	$s^4_J, \ldots, s^4_{I-1}$	V_2	$s^2_I, \ldots, s^2_\alpha$
		U_3	$s^6_J, \ldots, s^6_{I-1}$	V_3	$s^3_I, \ldots, s^3_\alpha$
		$\vdots$	$\vdots$		
		$U_{\frac{n-1}{2}}$	$s^{n-1}_J, \ldots, s^{n-1}_{I-1}$	$\vdots$	$\vdots$
				V_n	$s^n_I, \ldots, s^n_\alpha$

With this notation, algorithm $\boldsymbol{MF}_{\boldsymbol{Y},n}$ makes the following invocations of $\boldsymbol{Y}$. (Note that we include $\boldsymbol{Y}$'s random tape ρ in T.)

$$\boldsymbol{Y}(x,T,U_0,V_0),\ \boldsymbol{Y}(x,T,U_0,V_1),\ \boldsymbol{Y}(x,T,U_1,V_2),\ \boldsymbol{Y}(x,T,U_1,V_3),$$
$$\boldsymbol{Y}(x,T,U_2,V_4),\ \boldsymbol{Y}(x,T,U_2,V_5),\ \ldots,\ \boldsymbol{Y}(x,T,U_{\frac{n-1}{2}},V_{n-1}),\ \boldsymbol{Y}(x,T,U_{\frac{n-1}{2}},V_n)$$

Let $\mathcal{R}$ denote the set from which $\boldsymbol{Y}$ draws its coins at random. Then

$$\begin{aligned}
&\Pr\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_1,J_1)=(I,J)\wedge I\geq 1\wedge J\geq 1\,\right]\\
&= \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\Pr\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_1,J_1)=(I,J)=(i,j)\,\right]\\
&= \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\sum_{T}\frac{\Pr_{U_k,V_l}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_1,J_1)=(I,J)=(i,j)\,\right]}{|\mathcal{R}|\cdot|S|^{j-1}}\,. \qquad (7)
\end{aligned}$$

We compute the terms in the summation as follows:

$$\begin{aligned}
&\Pr_{U_k,V_l}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_1,J_1)=(I,J)=(i,j)\,\right]\\
&= \Pr_{U_k,V_l}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_3,J_3)=(I_2,J_2)=(i,j)\ \mid\ (I_1,J_1)=(I,J)=(i,j)\,\right]\cdot\\
&\qquad \Pr_{U_k,V_l}\left[\,(I_1,J_1)=(I,J)=(i,j)\,\right]\\
&= \Pr_{U_1,\ldots,U_{\frac{n-1}{2}},V_2,\ldots,V_n}\left[\,(I_n,J_n)=\cdots=(I_3,J_3)=(I_2,J_2)=(i,j)\ \mid\ (I_1,J_1)=(I,J)=(i,j)\,\right]\cdot\\
&\qquad \Pr_{U_0,V_0,V_1}\left[\,(I_1,J_1)=(I,J)=(i,j)\,\right]\\
&= \Pr_{U_1,\ldots,U_{\frac{n-1}{2}},V_2,\ldots,V_n}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_3,J_3)=(I_2,J_2)=(i,j)\,\right]\cdot\\
&\qquad \sum_{U_0}\frac{1}{|S|^{i-j}}\cdot\Pr_{V_0,V_1}\left[\,(I_1,J_1)=(I,J)=(i,j)\,\right]\\
&= \Pr_{U_1,\ldots,U_{\frac{n-1}{2}},V_2,\ldots,V_n}\left[\,(I_n,J_n)=\cdots=(I_4,J_4)=(i,j)\ \mid\ (I_3,J_3)=(I_2,J_2)=(i,j)\,\right]\cdot\\
&\qquad \Pr_{U_1,\ldots,U_{\frac{n-1}{2}},V_2,\ldots,V_n}\left[\,(I_3,J_3)=(I_2,J_2)=(i,j)\,\right]\cdot
\end{aligned}$$

$$
\begin{aligned}
& \sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0,V_1}\left[\,(I_1,J_1)=(i,j) \;\mid\; (I,J)=(i,j)\,\right] \cdot \Pr_{V_0,V_1}\left[\,(I,J)=(i,j)\,\right] \\
= \; & \Pr_{U_2,\ldots,U_{\frac{n-1}{2}},V_4,\ldots,V_n}\left[\,(I_n,J_n)=\cdots=(I_4,J_4)=(i,j) \;\mid\; (I_3,J_3)=(I_2,J_2)=(i,j)\,\right] \cdot \\
& \Pr_{U_1,V_2,V_3}\left[\,(I_3,J_3)=(I_2,J_2)=(i,j)\,\right] \cdot \\
& \sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_1}\left[\,(I_1,J_1)=(i,j) \;\mid\; (I,J)=(i,j)\,\right] \cdot \Pr_{V_0}\left[\,(I,J)=(i,j)\,\right] \\
= \; & \Pr_{U_2,\ldots,U_{\frac{n-1}{2}},V_4,\ldots,V_n}\left[\,(I_n,J_n)=\cdots=(I_4,J_4)=(i,j)\,\right] \cdot \\
& \sum_{U_1} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_2,V_3}\left[\,(I_3,J_3)=(I_2,J_2)=(i,j)\,\right] \cdot \\
& \sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_1}\left[\,(I_1,J_1)=(i,j)\,\right] \cdot \Pr_{V_0}\left[\,(I,J)=(i,j)\,\right] \\
= \; & \Pr_{U_2,\ldots,U_{\frac{n-1}{2}},V_4,\ldots,V_n}\left[\,(I_n,J_n)=\cdots=(I_6,J_6)=(i,j) \;\mid\; (I_5,J_5)=(I_4,J_4)=(i,j)\,\right] \cdot \\
& \Pr_{U_2,\ldots,U_{\frac{n-1}{2}},V_4,\ldots,V_n}\left[\,(I_5,J_5)=(I_4,J_4)=(i,j)\,\right] \cdot \\
& \sum_{U_1} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_2,V_3}\left[\,(I_3,J_3)=(i,j) \;\mid\; (I_2,J_2)=(i,j)\,\right] \cdot \Pr_{V_2,V_3}\left[\,(I_2,J_2)=(i,j)\,\right] \cdot \\
& \sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}\left[\,(I,J)=(i,j)\,\right]^2 \\
& \vdots \\
= \; & \Pr_{U_{\frac{n-1}{2}},V_{n-1},V_n}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=(i,j)\,\right] \cdot \\
& \left(\sum_{U_{\frac{n-3}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_{n-3}}\left[\,(I_{n-3},J_{n-3})=(i,j)\,\right]^2 \right) \cdot \;\cdots\; \cdot \left(\sum_{U_2} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_4}\left[\,(I_4,J_4)=(i,j)\,\right]^2 \right) \cdot \\
& \left(\sum_{U_1} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_2}\left[\,(I_2,J_2)=(i,j)\,\right]^2 \right) \cdot \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}\left[\,(I,J)=(i,j)\,\right]^2 \right) \\
= \; & \sum_{U_{\frac{n-1}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_{n-1},V_n}\left[\,(I_n,J_n)=(I_{n-1},J_{n-1})=(i,j)\,\right] \cdot \\
& \left(\sum_{U_{\frac{n-3}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_{n-3}}\left[\,(I_{n-3},J_{n-3})=(i,j)\,\right]^2 \right) \cdot \;\cdots\; \cdot \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}\left[\,(I,J)=(i,j)\,\right]^2 \right) \\
= \; & \sum_{U_{\frac{n-1}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_{n-1},V_n}\left[\,(I_n,J_n)=(i,j) \;\mid\; (I_{n-1},J_{n-1})=(i,j)\,\right] \cdot \Pr_{V_{n-1},V_n}\left[\,(I_{n-1},J_{n-1})=(i,j)\,\right] \cdot
\end{aligned}
$$

$$\left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n-1}{2}}$$

$$\begin{aligned}
&= \sum_{U_{\frac{n-1}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_n}[\,(I_n,J_n)=(i,j) \mid (I_{n-1},J_{n-1})=(i,j)\,] \cdot \Pr_{V_{n-1}}[\,(I_{n-1},J_{n-1})=(i,j)\,] \cdot \\
&\qquad \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n-1}{2}} \\
&= \sum_{U_{\frac{n-1}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_n}[\,(I_n,J_n)=(i,j)\,] \cdot \Pr_{V_{n-1}}[\,(I_{n-1},J_{n-1})=(i,j)\,] \cdot \\
&\qquad \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n-1}{2}} \\
&= \left(\sum_{U_{\frac{n-1}{2}}} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_{n-1}}[\,(I_{n-1},J_{n-1})=(i,j)\,]^2\right) \cdot \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n-1}{2}} \\
&= \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n+1}{2}} \qquad (8)
\end{aligned}$$

For each $i, j \in \{1, \ldots, \alpha\}$ and $T \in \mathcal{R} \times S^{j-1}$, we define a random variable $Y_{i,j,T} : S^{|i-j|} \to [0,1]$ (over the uniform distribution on its domain) via

$$Y_{i,j,T}(U_0) \quad = \quad \Pr\left[\, V_0 \xleftarrow{\$} S^{\alpha-i+1} \,;\, (I,J,\sigma) \leftarrow \mathbf{Y}(x,T,U_0,V_0) \;:\; (I,J)=(i,j) \,\right],$$

for all $U_0 \in S^{|i-j|}$. For each $i, j \in \{1, \ldots, \alpha\}$, we define a random variable $Z_{i,j} : \mathcal{R} \times S^{j-1} \to [0,1]$ (over the uniform distribution on its domain) via

$$Z_{i,j}(T) \quad = \quad \mathbf{E}[Y_{i,j,T}],$$

for all $T \in \mathcal{R} \times S^{j-1}$. Then, combining Equations (7) and (8), we have

$$\begin{aligned}
&\Pr[\,(I_n,J_n)=(I_{n-1},J_{n-1})=\cdots=(I_1,J_1)=(I,J) \wedge I \geq 1 \wedge J \geq 1\,] \\
&= \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\sum_{T} \frac{1}{|\mathcal{R}| \cdot |S|^{j-1}} \cdot \left(\sum_{U_0} \frac{1}{|S|^{i-j}} \cdot \Pr_{V_0}[\,(I,J)=(i,j)\,]^2\right)^{\frac{n+1}{2}} \\
&= \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\sum_{T} \frac{1}{|\mathcal{R}| \cdot |S|^{j-1}} \cdot \mathbf{E}\left[{Y_{i,j,T}}^2\right]^{\frac{n+1}{2}} \\
&\geq \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\sum_{T} \frac{1}{|\mathcal{R}| \cdot |S|^{j-1}} \cdot \mathbf{E}[Y_{i,j,T}]^{n+1} \qquad \text{(by Jensen's inequality with } f(x)=x^2\text{)} \\
&= \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha} \mathbf{E}\left[{Z_{i,j,T}}^{n+1}\right]
\end{aligned}$$

$$\begin{aligned}
&\geq \sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha} \mathbf{E}\left[Z_{i,j,T}\right]^{n+1} && \text{(by Jensen's inequality with } f(x)=x^{n+1}\text{)}\\
&\geq \frac{1}{\alpha^{2n}}\left(\sum_{i=1}^{\alpha}\sum_{j=1}^{\alpha}\mathbf{E}\left[Z_{i,j,T}\right]\right)^{n+1} && \text{(by Corollary C.4 with } n=\alpha^2, p=n+1, q=(n+1)/n\text{)}\\
&= \frac{1}{\alpha^{2n}}\cdot acc(x)^{n+1} && (9)
\end{aligned}$$

Combining Equations (5), (6) and (9), we get

$$\mathit{frk}(x) \;\;\geq\;\; \frac{acc(x)^{n+1}}{\alpha^{2n}} - \frac{n\cdot acc(x)}{|S|}\,.$$

Then, with the expectation taken over $x \xleftarrow{\$} \mathsf{IG}$, we have

$$\begin{aligned}
\mathit{frk} \;=\; \mathbf{E}\left[\mathit{frk}(x)\right] \;&\geq\; \mathbf{E}\left[\frac{acc(x)^{n+1}}{\alpha^{2n}} - \frac{n\cdot acc(x)}{|S|}\right] \;=\; \frac{\mathbf{E}\left[acc(x)^{n+1}\right]}{\alpha^{2n}} - \frac{n\cdot\mathbf{E}\left[acc(x)\right]}{|S|}\\
&\geq\; \frac{\mathbf{E}\left[acc(x)\right]^{n+1}}{\alpha^{2n}} - \frac{n\cdot\mathbf{E}\left[acc(x)\right]}{|S|} \;=\; acc\cdot\left(\frac{acc^n}{\alpha^{2n}} - \frac{n}{|S|}\right)\,.
\end{aligned}$$

The second line above follows from Lemma C.2 with $f(x) = x^{n+1}$. This completes the proof of Equation (3). We obtain Equation (4) from Equation (3) as follows.

$$\begin{aligned}
\mathit{frk} \;\geq\; \frac{acc^{n+1}}{\alpha^{2n}} - \frac{n\cdot acc}{|S|} \;&\geq\; \frac{acc^{n+1}}{\alpha^{2n}} - \frac{n}{|S|}\\
\sqrt[n+1]{\alpha^{2n}\cdot\mathit{frk} + \frac{n\cdot\alpha^{2n}}{|S|}} \;&\geq\; acc\\
\sqrt[n+1]{\alpha^{2n}\cdot\mathit{frk}} \;+\; \sqrt[n+1]{\frac{n\cdot\alpha^{2n}}{|S|}} \;&\geq\; acc\,.
\end{aligned}$$

The last equation follows from the fact that $\sqrt[n+1]{a} + \sqrt[n+1]{b} \geq \sqrt[n+1]{a+b}$ for any real numbers $a, b \geq 0$. ▌

We now prove Theorem 6.2. As is usual in the random-oracle model, the hash functions G, R, H used in the scheme are assumed to behave as random oracles, i.e., they are assumed to be chosen independently at random from all functions $f : \{0,1\}^* \to \mathbb{Z}_q$, and all parties (including the adversary) are given access to these oracles.

Let $\boldsymbol{A}$ be an adversary against TS that makes at most q_G queries to random oracle G, q_R queries to random oracle R, q_H queries to random oracle H, q_d requests to be designated by user 1, q_{sd} self-delegation requests, q_s standard signature queries, and q_p proxy signature queries. Without loss of generality, we assume that the adversary does not repeat any random-oracle queries (it can just store the responses in a table). Fix $\kappa \in \mathbb{N}$. Consider experiment $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{TS},\boldsymbol{A}}(\kappa)$. Recall that $\mathbf{CS}$ contains the messages for which $\boldsymbol{A}$ can produce proxy signatures by user 1 on behalf of user 1 using compromised self-delegated proxy signing keys, and $\mathbf{DU}$ contains the identities of the users designated by user 1 together with the descriptions of the message spaces for which they are designated. We define the following events associated to experiment $\mathbf{Exp}^{\text{ps-uf}}_{\mathsf{TS},\boldsymbol{A}}(\kappa)$.

E_1 : $\boldsymbol{A}$'s forgery is of the form (M, σ), where $\mathcal{V}(pk_1, M, \sigma) = 1$, and M was not queried to oracle $\mathcal{O}_{\mathcal{S}_\mathsf{T}}(sk_1, \cdot)$

E_2 : $\boldsymbol{A}$'s forgery is of the form $(M, p\sigma, pk_i)$, where $i \neq 1$, $\mathcal{PV}(pk_i, M, p\sigma) = 1$, $\mathcal{ID}(p\sigma) = 1$, and no valid query (i, l, M), for $l \in \mathbb{N}$, was made to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$

E_3 : $\boldsymbol{A}$'s forgery is of the form $(M, (1, \omega, Y, pk_1', (V, z)), pk_1)$, where $\mathcal{PV}(pk_1, M, (1, \omega, Y, pk_1', (V, z))) = 1$, no valid query $(1, l, M)$, for $l \in \mathbb{N}$, was made to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, $M \notin \mathbf{CS}$, and $\boldsymbol{A}$ did not make a self-delegation request that was answered with a Schnorr signature (Y, s) for message $0||pk_1||1||pk_1'||\omega$

E_4 : $\boldsymbol{A}$'s forgery is of the form $(M, (1, \omega, Y, pk_1', (V, z)), pk_1)$, where $\mathcal{PV}(pk_1, M, (1, \omega, Y, pk_1', (V, z))) = 1$, no valid query $(1, l, M)$, for $l \in \mathbb{N}$, was made to oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, $M \notin \mathbf{CS}$, and $\boldsymbol{A}$ made a self-delegation request that was answered with a Schnorr signature (Y, s) for message $0||pk_1||1||pk_1'||\omega$

E_5 : $\boldsymbol{A}$'s forgery is of the form $(M, p\sigma, pk_1)$, where $\mathcal{PV}(pk_1, M, p\sigma) = 1$, and for all message spaces ω with $(\mathcal{ID}(p\sigma), \omega) \in \mathbf{DU}$ it holds that $M \notin \omega$

It follows from Definition 3.2 that $\Pr\left[\, \mathbf{Exp}^{\text{ps-uf}}_{\mathsf{TS},\boldsymbol{A}}(\kappa) = 1 \,\right] = \Pr[\, E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \,]$. Therefore,

$$\mathbf{Adv}^{\text{ps-uf}}_{\mathsf{TS},\boldsymbol{A}}(\kappa) \;\; \leq \;\; \Pr[\, E_1 \,] + \Pr[\, E_2 \,] + \Pr[\, E_3 \,] + \Pr[\, E_4 \,] + \Pr[\, E_5 \,]\,. \tag{10}$$

We will construct adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ $\boldsymbol{E}$, and $\boldsymbol{F}$ against the discrete-logarithm parameter generator $\mathcal{G}_{\text{dl}}$ underlying TS such that

$$\begin{aligned}\Pr[\, E_1 \,] \;\leq\; & \sqrt{q_G \cdot \mathbf{Adv}^{\text{dl}}_{\mathcal{G}_{\text{dl}},\boldsymbol{B}}(\kappa)} + \\ & \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + q_G + 1}{q},\end{aligned} \tag{11}$$

$$\begin{aligned}\Pr[\, E_2 \,] \;\leq\; & \sqrt[4]{(q_R + q_H)^6 \cdot \mathbf{Adv}^{\text{dl}}_{\mathcal{G}_{\text{dl}},\boldsymbol{C}}(\kappa)} + \sqrt[4]{\frac{3(q_R + q_H)^6}{q}} + \\ & \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q},\end{aligned} \tag{12}$$

$$\begin{aligned}\Pr[\, E_3 \,] \;\leq\; & \sqrt[6]{(q_G + q_H)^{10} \cdot \mathbf{Adv}^{\text{dl}}_{\mathcal{G}_{\text{dl}},\boldsymbol{D}}(\kappa)} + \sqrt[6]{\frac{(q_G + q_H)^{10}}{q}} + \sqrt[6]{\frac{5(q_G + q_H)^{10}}{q}} + \\ & \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q},\end{aligned} \tag{13}$$

$$\Pr[\, E_4 \,] \;\leq\; q_{sd} \cdot \sqrt{\delta \cdot \mathbf{Adv}^{\text{dl}}_{\mathcal{G}_{\text{dl}},\boldsymbol{E}}(\kappa)} + q_{sd} \cdot \sqrt{\frac{\delta}{q}} + \frac{\delta \cdot q_{sd} + 3}{q}, \tag{14}$$

$$\begin{aligned}\Pr[\, E_5 \,] \;\leq\; & \sqrt[6]{(q_G + q_H)^{10} \cdot \mathbf{Adv}^{\text{dl}}_{\mathcal{G}_{\text{dl}},\boldsymbol{F}}(\kappa)} + \sqrt[6]{\frac{(q_G + q_H)^{10}}{q}} + \sqrt[6]{\frac{5(q_G + q_H)^{10}}{q}} + \\ & \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q}.\end{aligned} \tag{15}$$

Combining these five equations with Equation (10), we obtain Equation (2). We proceed to define adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, $\boldsymbol{E}$, and $\boldsymbol{F}$.

Let $\alpha = q_G$ and $S = \mathbb{Z}_q$. We first define an algorithm $\boldsymbol{Y}$ that given inputs a public key (p, q, g, X) and $s_1, \ldots, s_\alpha \in S$, returns a pair (I, σ) consisting of an integer $0 \leq I \leq \alpha$ and a string σ. Then we use the forking algorithm $\boldsymbol{F_Y}$ associated to $\boldsymbol{Y}$ to construct adversary $\boldsymbol{B}$ against $\mathcal{G}_{\mathrm{dl}}$.

$\boldsymbol{Y}$ sets $n = 1$, $pk_1 = X$, and $j = 0$; creates empty sets $\bar{S}$, S_1, $\mathbf{DU}$, and $\mathbf{CS}$; creates empty arrays $\mathbf{skp}_1$, GT, RT, and HT; chooses some randomness for $\boldsymbol{A}$; and runs $\boldsymbol{A}$ on input pk_1 with this randomness. It then answers the requests and queries made by $\boldsymbol{A}$ as follows.

1. If $\boldsymbol{A}$ makes a query $1||M||Y$ to random oracle G, then $\boldsymbol{Y}$ checks if $GT[1||M||Y]$ is defined. If not, it increments j and sets $M_j||Y_j = M||Y$ and $GT[1||M||Y] = s_j$. Then it returns $GT[1||M||Y]$ to $\boldsymbol{A}$.
2. If $\boldsymbol{A}$ makes a query $0||M$ to random oracle G, then $\boldsymbol{Y}$ checks if $GT[0||M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[0||M] = c$. Then it returns $GT[0||M]$ to $\boldsymbol{A}$.
3. If $\boldsymbol{A}$ makes a query M to random oracle R, then $\boldsymbol{Y}$ checks if $RT[M]$ is defined. If not, it picks a random $r \in \mathbb{Z}_q$ and sets $RT[M] = r$. Then it returns $RT[M]$ to $\boldsymbol{A}$.
4. If $\boldsymbol{A}$ makes a query M to random oracle H, then $\boldsymbol{Y}$ checks if $HT[M]$ is defined. If not, it picks a random $h \in \mathbb{Z}_q$ and sets $HT[M] = h$. Then it returns $HT[M]$ to $\boldsymbol{A}$.
5. If $\boldsymbol{A}$ requests to register a public key pk for user $n + 1$, then $\boldsymbol{Y}$ increments n, sets $pk_n = pk$, $S_n = \emptyset$ and creates an empty array $\mathbf{skp}_n$.
6. If $\boldsymbol{A}$ requests to interact with user 1 running $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, for some $i \in \{2, \ldots, n\}$, and play the role of user i running $\mathcal{P}(pk_i, sk_i, pk_1)$, then $\boldsymbol{Y}$ creates an appropriate message space description ω, sets $\mathbf{DU} = \mathbf{DU} \cup \{(i, \omega)\}$, and performs the following operations:
   - Pick random $c \in \mathbb{Z}_q$, $s \in \mathbb{Z}_q$
   - Compute commitment $Y = g^s \cdot pk_1^{-c} \bmod p$
   - If $GT[0||pk_1||i||pk_i||\omega||Y]$ is defined, set $\mathsf{bad} = \mathsf{true}$
   - Set $GT[0||pk_1||i||pk_i||\omega||Y] = c$
   - Return ω, (Y, s) to $\boldsymbol{A}$

   Thus, $\boldsymbol{Y}$ simulates standard signing by user 1 for message $0||pk_1||i||pk_i||\omega$. It is easy to see that the simulated signature (Y, s) has the same distribution as a real Schnorr signature for that message. Therefore, the signature returned to adversary $\boldsymbol{A}$ has the same distribution as a signature generated by user 1 during proxy designation.
7. If $\boldsymbol{A}$ requests to interact with user 1 running $\mathcal{P}(pk_1, sk_1, pk_i)$, for some $i \in \{2, \ldots, n\}$, and play the role of user i running $\mathcal{D}(pk_i, sk_i, 1, pk_1, \omega)$, when $\boldsymbol{A}$ outputs ω, (Y, s), $\boldsymbol{Y}$ performs the following operations:
   - If $GT[0||pk_i||1||pk_1||\omega||Y]$ is defined, set $c = GT[0||pk_i||1||pk_1||\omega||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[0||pk_i||1||pk_1||\omega||Y] = c$.
   - Verify that (Y, s) is a valid signature for message $0||pk_i||1||pk_1||\omega$ with respect to public key pk_i (i.e., check that $g^s \equiv Y \cdot pk_i^c \pmod{p}$). If so, store ω, Y, s in the last unoccupied position of $\mathbf{skp}_i$. Otherwise, abort.
8. If $\boldsymbol{A}$ requests that user 1 run the designation protocol with itself for ω, then $\boldsymbol{Y}$ creates a new key pair (pk_1', sk_1') by selecting $sk_1' \in \mathbb{Z}_q$ at random and setting $pk_1' = g^{sk_1'} \bmod p$, and performs the following operations:
   - Pick a random $c \in \mathbb{Z}_q$, $s \in \mathbb{Z}_q$
   - Compute commitment $Y = g^s \cdot pk_1^{-c} \bmod p$
   - If $GT[0||pk_1||1||pk_1'||\omega||Y]$ is defined, set $\mathsf{bad} = \mathsf{true}$
   - Set $GT[0||pk_1||1||pk_1'||\omega||Y] = c$
   - If $RT[pk_1||1||pk_1'||\omega||Y||c]$ is defined, set $r = RT[pk_1||1||pk_1'||\omega||Y||c]$. Otherwise, pick a random $r \in \mathbb{Z}_q$ and set $RT[pk_1||1||pk_1'||\omega||Y||c] = r$.

- Set $t = r \cdot sk'_1 + s \bmod q$
- Set $skp = (pk_1||1||pk'_1||\omega, Y, t)$
- Store (skp, ω) in the last unoccupied position of $\mathbf{skp}_1$
- Return $\omega, (Y, s)$ to $\boldsymbol{A}$

Here $\boldsymbol{Y}$ simulates standard signing by user 1 for message $0||pk_1||1||pk'_1||\omega$. Using the signature obtained, it computes a correct proxy signing key for user 1.

9. If $\boldsymbol{A}$ requests to see $\mathbf{skp}_1[l]$ for some $l \in \mathbb{N}$, then if $\mathbf{skp}_1[l]$ contains a proxy signing key and message space pair (skp, ω), $\boldsymbol{Y}$ sets $\mathbf{CS} = \mathbf{CS} \cup \omega$ and returns skp to $\boldsymbol{A}$; otherwise, $\boldsymbol{Y}$ returns $\perp$ to $\boldsymbol{A}$.

10. If $\boldsymbol{A}$ queries its oracle $\mathcal{O}_{\mathcal{S}_{\mathsf{T}}}(sk_1, \cdot)$ with a message M, then $\boldsymbol{Y}$ performs the following operations:
   - Pick a random $c \in \mathbb{Z}_q$, $s \in \mathbb{Z}_q$
   - Compute commitment $Y = g^s \cdot pk_1^{-c} \bmod p$
   - If $GT[1||M||Y]$ is defined, set $\mathsf{bad} = \mathsf{true}$
   - Set $GT[1||M||Y] = c$
   - Set $\bar{S} = \bar{S} \cup \{M\}$
   - Return (Y, s) to $\boldsymbol{A}$

   $\boldsymbol{Y}$ simulates standard signing by user 1 for message $1||M$.

11. If $\boldsymbol{A}$ makes a query (i, l, M), where $i \in \{2, \ldots, n\}$, $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u \in [n]}, \cdot, \cdot, \cdot)$, then $\boldsymbol{Y}$ responds as follows. If $\mathbf{skp}_i[l]$ is not defined, then it returns $\perp$ to $\boldsymbol{A}$. Otherwise, it parses $\mathbf{skp}_i[l]$ as ω_l, Y_l, s_l, and performs the following operations:
   - Pick a random $h \in \mathbb{Z}_q$, $s \in \mathbb{Z}_q$
   - Set $c = GT[0||pk_i||1||pk_1||\omega_l||Y_l]$
   - If $RT[pk_i||1||pk_1||\omega_l||Y_l||c]$ is defined, set $r = RT[pk_i||1||pk_1||\omega_l||Y_l||c]$. Otherwise, pick a random $r \in \mathbb{Z}_q$ and set $RT[pk_i||1||pk_1||\omega_l||Y_l||c] = r$.
   - Compute proxy public key $pkp = pk_1^r \cdot Y_l \cdot pk_i^c \bmod p$
   - Compute commitment $Y \leftarrow g^s \cdot pkp^{-h} \bmod p$
   - If $HT[0||M||pk_i||1||pk_1||\omega_l||Y_l||r||Y]$ is defined, set $\mathsf{bad} = \mathsf{true}$
   - Set $HT[0||M||pk_i||1||pk_1||\omega_l||Y_l||r||Y] = h$
   - Set $S_i = S_i \cup \{M\}$
   - Return $(1, \omega_l, Y_l, pk_1, (Y, s))$ to $\boldsymbol{A}$

   Thus, $\boldsymbol{Y}$ simulates proxy signing by user 1 on behalf of user i using the l-th proxy signing key. It is easy to see that the simulated signature (Y, s) has the same distribution as a real Schnorr signature for message $0||M||pk_i||1||pk_1||\omega_l||Y_l||r$. Therefore, the signature returned to adversary $\boldsymbol{A}$ has the same distribution as a signature returned by oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u \in [n]}, \cdot, \cdot, \cdot)$.

12. If $\boldsymbol{A}$ makes a query $(1, l, M)$, where $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u \in [n]}, \cdot, \cdot, \cdot)$, then $\boldsymbol{Y}$ responds as follows. If $\mathbf{skp}_1[l]$ is not defined, then it returns $\perp$ to $\boldsymbol{A}$. Otherwise, it parses $\mathbf{skp}_1[l]$ as $((pk_1||1||pk_1^l||\omega_l, Y_l, t_l), \omega)$, and performs the following operations:
   - Pick a random $y \in \mathbb{Z}_q$
   - Compute commitment $Y \leftarrow g^y \bmod p$
   - Set $c = GT[0||pk_1||1||pk_1^l||\omega_l||Y_l]$
   - Set $r = RT[pk_1||1||pk_1^l||\omega_l||Y_l||c]$
   - If $HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y]$ is defined, set $h = HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y]$. Otherwise, pick a random $h \in \mathbb{Z}_q$ and set $HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y] = h$.
   - Set $s = y + t_l \cdot h \bmod q$

- Set $S_1 = S_1 \cup \{M\}$
- Return $(1, \omega_l, Y_l, pk_1^l, (Y, s))$ to $\boldsymbol{A}$

$\boldsymbol{Y}$ computes a proxy signature by user 1 on behalf of herself using the l-th proxy signing key $(pk_1||1||pk_1^l||\omega_l, Y_l, t_l)$. The signature returned to adversary $\boldsymbol{A}$ is thus identical to the signature returned by oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$.

Until $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. If $\boldsymbol{A}$'s forgery is not of the form (M, σ), then $\boldsymbol{Y}$ aborts. Otherwise, $\boldsymbol{Y}$ performs the following operations:

- Parse σ as (V, z)
- If $\boldsymbol{A}$ did not make query $1||M||V$ to random oracle G then set $\mathsf{bad} = \mathsf{true}$. Otherwise, set $c = GT[1||M||V]$.
- If $\mathsf{bad} = \mathsf{true}$ or $g^z \not\equiv V \cdot pk_1^c \pmod p$ or $M \in \bar{S}$, then return $(0, \varepsilon)$
- Let i be such that $M_i||Y_i = M||V$
- Return $(i, (z, s_i))$

Let IG be the algorithm that runs $\mathcal{K}_{\mathsf{T}}(1^\kappa)$ to obtain (pk, sk) and returns $pk = (p, q, g, X)$. Let

$$acc \;=\; \Pr\Big[\, pk \xleftarrow{\$} \mathsf{IG}\, ;\, s_1, \ldots, s_\alpha \xleftarrow{\$} \mathbb{Z}_q\, ;\, (I, \sigma) \xleftarrow{\$} \boldsymbol{Y}(pk, s_1, \ldots, s_\alpha) \;:\; I \geq 1 \,\Big],$$

as in Lemma C.1. Assume that event E_1 occurs and $\mathsf{bad} \neq \mathsf{true}$. Then when $\boldsymbol{A}$ makes query $1||M||V$ to random oracle G, $GT[1||M||V]$ is undefined and gets set to s_i for some i such that $1 \leq i \leq \alpha$. Therefore, $\boldsymbol{Y}$ returns $(i, (z, s_i))$ for some $i \geq 1$. Thus,

$$\begin{aligned} acc &\geq \Pr[\, E_1 \wedge \mathsf{bad} \neq \mathsf{true}\,] = \Pr[\, E_1\,] \cdot \Pr[\, \mathsf{bad} \neq \mathsf{true} \mid E_1\,] \\ &\geq \Pr[\, E_1\,] - \Pr[\, \mathsf{bad} = \mathsf{true} \mid E_1\,] \\ &\geq \Pr[\, E_1\,] - \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 1}{|\mathbb{Z}_q|} \end{aligned}$$

Let $\boldsymbol{F_Y}$ be the forking algorithm associated to $\boldsymbol{Y}$ as per Lemma C.1. Then we define adversary $\boldsymbol{B}$ against discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ as follows.

```
Adversary $\boldsymbol{B}(p, q, g, X)$
    $pk \leftarrow (p, q, g, X)$ ; $(b, \sigma, \sigma') \xleftarrow{\$} \boldsymbol{F_Y}(pk)$
    If ( $b = 0$ ) then return 0
    Parse $\sigma$ as $(z, s)$ and $\sigma'$ as $(z', s')$
    Return $(z - z')(s - s')^{-1} \bmod q$
```

We claim that if $b = 1$ then $\boldsymbol{B}$ computes the discrete logarithm of X. To justify this claim, consider the definitions of $\boldsymbol{Y}$ and $\boldsymbol{F_Y}$. If $b = 1$ then there exist coins ρ for $\boldsymbol{Y}$, $i \geq 1$ and $s_1, \ldots, s_\alpha, s_i', \ldots, s_\alpha' \in \mathbb{Z}_q$ with $s' = s_i' \neq s_i = s$ such that 1) in the execution of $\boldsymbol{Y}(pk, s_1, \ldots, s_\alpha; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(M, (V, z))$ with $M||V = M_i||Y_i$ and $GT[1||M||V] = s_i$, and 2) in the execution of $\boldsymbol{Y}(pk, s_1, \ldots, s_{i-1}, s_i', \ldots, s_\alpha'; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(M', (V', z'))$ with $M'||V' = M_i||Y_i$ and $GT[1||M'||V'] = s_i'$. It follows that $M' = M$, $V' = V$, $g^z \equiv V \cdot X^s \pmod p$, and $g^{z'} \equiv V \cdot X^{s'} \pmod p$. Thus, $g^{(z-z')(s-s')^{-1}} \equiv X \pmod p$, as desired.

Let *frk* be defined as in Lemma C.1. Applying this lemma, we have

$$\begin{aligned} &\Pr[\, E_1\,] \\ &\quad\leq\; acc + \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 1}{q} \\ &\quad\leq\; \sqrt{\alpha \cdot \mathit{frk}} + \frac{\alpha}{q} + \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 1}{q} \end{aligned}$$

$$\leq \sqrt{q_G \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{B}}(\kappa)} + \frac{q_d(q_d-1+q_G)+q_{sd}(q_{sd}-1+q_G)+q_s(q_s-1+q_G)+q_p(q_p-1+q_H)+q_G+1}{q}\,.$$

This proves Equation (11).

Let $\beta = q_R + q_H$ and $S = \mathbb{Z}_q$. Next, we define an algorithm $\boldsymbol{Z}$ that given inputs a public key (p, q, g, X) and $s_1, \ldots, s_\beta \in S$, returns a triple (I, J, σ) consisting of two integers $0 \leq J < I \leq \beta$ and a string σ. Then we use the multiple-forking algorithm $\boldsymbol{MF}_{\boldsymbol{Z},3}$ associated to $\boldsymbol{Z}$ and 3 to construct adversary $\boldsymbol{C}$ against $\mathcal{G}_{\mathrm{dl}}$.

$\boldsymbol{Z}$ is very similar to algorithm $\boldsymbol{Y}$ defined above. It makes the same initializations: $n = 1$, $pk_1 = X$, and $j = 0$; creates empty sets $\bar{S}$, S_1, $\mathbf{DU}$, and $\mathbf{CS}$; creates empty arrays $\mathbf{skp}_1$, GT, RT, and HT; chooses some randomness for $\boldsymbol{A}$; and runs $\boldsymbol{A}$ on input pk_1 with this randomness. It then answers the requests and queries made by $\boldsymbol{A}$ exactly as $\boldsymbol{Y}$ does except for the differences specified below. These are identified by the number(s) of the corresponding step(s) in $\boldsymbol{Y}$.

1, 2. If $\boldsymbol{A}$ makes a query M to random oracle G, then $\boldsymbol{Z}$ checks if $GT[M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[M] = c$. Then it returns $GT[M]$ to $\boldsymbol{A}$.

3. If $\boldsymbol{A}$ makes a query $pk_i||1||pk_1||\omega||Y||c$ to random oracle R, where $i \in \{2, \ldots, n\}$, then $\boldsymbol{Z}$ checks if $RT[pk_i||1||pk_1||\omega||Y||c]$ is defined. If not, it increments j and sets $pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||c_j = pk_i||1||pk_1||\omega||Y||c$ and $RT[pk_i||1||pk_1||\omega||Y||c] = s_j$. Then it returns $RT[pk_i||1||pk_1||\omega||Y||c]$ to $\boldsymbol{A}$.

3. If $\boldsymbol{A}$ makes a query M that cannot be parsed as $pk_i||1||pk_1||\omega||Y||c$, for some $i \in \{2, \ldots, n\}$, to random oracle R, then $\boldsymbol{Z}$ checks if $RT[M]$ is defined. If not, it picks a random $r \in \mathbb{Z}_q$ and sets $RT[M] = r$. Then it returns $RT[M]$ to $\boldsymbol{A}$.

4. If $\boldsymbol{A}$ makes a query $0||M||pk_i||1||pk_1||\omega||Y||r||V$ to random oracle H, where $i \in \{2, \ldots, n\}$, then $\boldsymbol{Z}$ checks if $HT[0||M||pk_i||1||pk_1||\omega||Y||r||V]$ is defined. If not, it increments j and sets $0||M_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_i||1||pk_1||\omega||Y||r||V$ and $HT[0||M||pk_i||1||pk_1||\omega||Y||r||V] = s_j$. Then it returns $HT[0||M||pk_i||1||pk_1||\omega||Y||r||V]$ to $\boldsymbol{A}$.

4. If $\boldsymbol{A}$ makes a query M' that cannot be parsed as $0||M||pk_i||1||pk_1||\omega||Y||r||V$, for some $i \in \{2, \ldots, n\}$, to random oracle H, then $\boldsymbol{Z}$ checks if $HT[M']$ is defined. If not, it picks a random $h \in \mathbb{Z}_q$ and sets $HT[M'] = h$. Then it returns $HT[M']$ to $\boldsymbol{A}$.

Until $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. If $\boldsymbol{A}$'s forgery is not of the form $(M, p\sigma, pk_i)$ for some $i \in \{2, \ldots, n\}$, where $\mathcal{ID}(p\sigma) = 1$, then $\boldsymbol{Z}$ aborts. Otherwise, $\boldsymbol{Z}$ performs the following operations:

- Parse $p\sigma$ as $(1, \omega, Y, pk_1, (V, z))$
- If $\boldsymbol{A}$ did not make the following queries in the order given, then set $\mathsf{bad} = \mathsf{true}$.
    - $0||pk_i||1||pk_1||\omega||Y$ to random oracle G,
    - $pk_i||1||pk_1||\omega||Y||c$, where c is the response to the G-query above, to random oracle R,
    - $0||M||pk_i||1||pk_1||\omega||Y||r||V$, where r is the response to the R-query above, to H

  Otherwise, set $c = GT[0||pk_i||1||pk_1||\omega||Y]$, $r = RT[pk_i||1||pk_1||\omega||Y||c]$, and $h = HT[0||M||pk_i||1||pk_1||\omega||Y||r||V]$.
- If $\mathsf{bad} \neq \mathsf{true}$, compute proxy public key $pkp = pk_1^r \cdot Y \cdot pk_i^c \bmod p$
- If $\mathsf{bad} = \mathsf{true}$ or $g^z \not\equiv V \cdot pkp^h \pmod{p}$ or $M \in S_i$, then return $(0, 0, \varepsilon)$
- Let j be such that $0||M_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_i||1||pk_1||\omega||Y||r||V$, and k such that $pk_{i,k}||1||pk_{1,k}||\omega_k||Y_k||c_k = pk_i||1||pk_1||\omega||Y||c$
- Return $(j, k, (z, s_j, s_k))$

Let IG be the algorithm that runs $\mathcal{K}_{\mathsf{T}}(1^{\kappa})$ to obtain (pk, sk) and returns $pk = (p, q, g, X)$. Let

$$acc \;=\; \Pr\Big[\, pk \xleftarrow{\$} \mathsf{IG}\ ;\ s_1, \ldots, s_\beta \xleftarrow{\$} \mathbb{Z}_q\ ;\ (I, J, \sigma) \xleftarrow{\$} \boldsymbol{Z}(pk, s_1, \ldots, s_\beta)\ :\ I \geq 1 \wedge J \geq 1 \,\Big],$$

as in Lemma C.5. Assume that event E_2 occurs and $\mathsf{bad} \neq \mathsf{true}$. Then when $\boldsymbol{A}$ makes query $pk_i||1||pk_1||\omega||Y||c$ to random oracle R, $RT[pk_i||1||pk_1||\omega||Y||c]$ is undefined and gets set to s_k for some k such that $1 \leq k \leq \beta$. In addition, when $\boldsymbol{A}$ makes query $0||M||pk_i||1||pk_1||\omega||Y||r||V$ to random oracle H, $HT[0||M||pk_i||1||pk_1||\omega||Y||r||V]$ is undefined and gets set to s_j for some $j > k$ such that $1 \leq j \leq \beta$. Therefore, $\boldsymbol{Z}$ returns $(j, k, (z, s_j, s_k))$ for some $j > k \geq 1$. Thus,

$$\begin{aligned} acc &\geq \Pr[\, E_2 \wedge \mathsf{bad} \neq \mathsf{true} \,] \geq \Pr[\, E_2 \,] - \Pr[\, \mathsf{bad} = \mathsf{true} \mid E_2 \,] \\ &\geq \Pr[\, E_2 \,] - \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{|\mathbb{Z}_q|} \end{aligned}$$

Let $\boldsymbol{MF}_{\boldsymbol{Z},3}$ be the multiple-forking algorithm associated to $\boldsymbol{Z}$ and 3 as per Lemma C.5. Then we define adversary $\boldsymbol{C}$ against discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ as follows.

```
Adversary $\boldsymbol{C}(p, q, g, X)$
    $pk \leftarrow (p, q, g, X)$ ; $(b, \mathsf{results}) \xleftarrow{\$} \boldsymbol{MF}_{\boldsymbol{Z},3}(pk)$
    If ( $b = 0$ ) then return 0
    Parse $\mathsf{results}[0]$ as $(z, h, r)$, $\mathsf{results}[1]$ as $(\hat{z}, \hat{h}, \hat{r})$, $\mathsf{results}[2]$ as $(\bar{z}, \bar{h}, \bar{r})$, $\mathsf{results}[3]$ as $(\dot{z}, \dot{h}, \dot{r})$
    Return $\Big((z - \hat{z})(h - \hat{h})^{-1} - (\bar{z} - \dot{z})(\bar{h} - \dot{h})^{-1}\Big) \cdot (r - \bar{r})^{-1} \bmod q$
```

We claim that if $b = 1$ then $\boldsymbol{C}$ computes the discrete logarithm of X. To justify this claim, consider the definitions of $\boldsymbol{Z}$ and $\boldsymbol{MF}_{\boldsymbol{Z},3}$. If $b = 1$ then there exist coins ρ for $\boldsymbol{Z}$, $j > k \geq 1$ and $s_1, \ldots, s_\beta, s_j^1, \ldots, s_\beta^1, s_k^2, \ldots, s_\beta^2, s_j^3, \ldots, s_\beta^3 \in \mathbb{Z}_q$ with $\hat{h} = s_j^1 \neq s_j = h$, $\hat{r} = s_k = r$, $\bar{r} = s_k^2 \neq s_k = r$, $\dot{h} = s_j^3 \neq s_j^2 = \bar{h}$, and $\dot{r} = s_k^2 = \bar{r}$, such that

1) in the execution of $\boldsymbol{Z}(pk, s_1, \ldots, s_\beta; \rho)$, adversary $\boldsymbol{A}$ outputs a valid forgery $(M, (1, \omega, Y, pk_1, (V, z)), pk_i)$ with $c = GT[0||pk_i||1||pk_1||\omega||Y]$, $r = RT[pk_i||1||pk_1||\omega||Y||c] = s_k$, $h = HT[0||M||pk_i||1||pk_1||\omega||Y||r||V] = s_j$, $pk_i||1||pk_1||\omega||Y||c = pk_{i,k}||1||pk_{1,k}||\omega_k||Y_k||c_k$, and $0||M||pk_i||1||pk_1||\omega||Y||r||V = 0||M_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||r_j||V_j$,

2) in the execution of $\boldsymbol{Z}(pk, s_1, \ldots, s_{j-1}, s_j^1, \ldots, s_\beta^1; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\hat{M}, (1, \hat{\omega}, \hat{Y}, p\hat{k}_1, (\hat{V}, \hat{z})), p\hat{k}_i)$ with $\hat{c} = GT[0||p\hat{k}_i||1||p\hat{k}_1||\hat{\omega}||\hat{Y}]$, $\hat{r} = RT[p\hat{k}_i||1||p\hat{k}_1||\hat{\omega}||\hat{Y}||\hat{c}] = s_k$, $\hat{h} = HT[0||\hat{M}||p\hat{k}_i||1||p\hat{k}_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V}] = s_j^1$, $p\hat{k}_i||1||p\hat{k}_1||\hat{\omega}||\hat{Y}||\hat{c} = pk_{i,k}||1||pk_{1,k}||\omega_k||Y_k||c_k$, and $0||\hat{M}||p\hat{k}_i||1||p\hat{k}_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V} = 0||M_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||r_j||V_j$,

3) in the execution of $\boldsymbol{Z}(pk, s_1, \ldots, s_{k-1}, s_k^2, \ldots, s_\beta^2; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\bar{M}, (1, \bar{\omega}, \bar{Y}, p\bar{k}_1, (\bar{V}, \bar{z})), p\bar{k}_i)$ with $\bar{c} = GT[0||p\bar{k}_i||1||p\bar{k}_1||\bar{\omega}||\bar{Y}]$, $\bar{r} = RT[p\bar{k}_i||1||p\bar{k}_1||\bar{\omega}||\bar{Y}||\bar{c}] = s_k^2$, $\bar{h} = HT[0||\bar{M}||p\bar{k}_i||1||p\bar{k}_1||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V}] = s_j^2$, $p\bar{k}_i||1||p\bar{k}_1||\bar{\omega}||\bar{Y}||\bar{c} = pk_{i,k}||1||pk_{1,k}||\omega_k||Y_k||c_k$, and $0||\bar{M}||p\bar{k}_i||1||p\bar{k}_1||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V} = 0||\bar{M}_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$, and

4) in the execution of $\boldsymbol{Z}(pk, s_1, \ldots, s_{k-1}, s_k^2, \ldots, s_{j-1}^2, s_j^3, \ldots, s_\beta^3; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\dot{M}, (1, \dot{\omega}, \dot{Y}, p\dot{k}_1, (\dot{V}, \dot{z})), p\dot{k}_i)$ with $\dot{c} = GT[0||p\dot{k}_i||1||p\dot{k}_1||\dot{\omega}||\dot{Y}]$, $\dot{r} = RT[p\dot{k}_i||1||p\dot{k}_1||\dot{\omega}||\dot{Y}||\dot{c}] = s_k^2$, $\dot{h} = HT[0||\dot{M}||p\dot{k}_i||1||p\dot{k}_1||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V}] = s_j^3$, $p\dot{k}_i||1||p\dot{k}_1||\dot{\omega}||\dot{Y}||\dot{c} = pk_{i,k}||1||pk_{1,k}||\omega_k||Y_k||c_k$, and $0||\dot{M}||p\dot{k}_i||1||p\dot{k}_1||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V} = 0||\bar{M}_j||pk_{i,j}||1||pk_{1,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$.

From 1) and 2), it follows that $\hat{M} = M$, $p\hat{k}_i = pk_i$, $p\hat{k}_1 = pk_1$, $\hat{\omega} = \omega$, $\hat{Y} = Y$, $\hat{r} = r$, $\hat{V} = V$, $\hat{c} = c$, $g^z \equiv V \cdot (X^r \cdot Y \cdot pk_i^c)^h \pmod{p}$, and $g^{\hat{z}} \equiv V \cdot (X^r \cdot Y \cdot {pk_i}^c)^{\hat{h}} \pmod{p}$. Since $\hat{h} \neq h$, $(h - \hat{h})^{-1}$ is well-defined. Thus,

$$g^{(z - \hat{z})(h - \hat{h})^{-1}} \equiv X^r \cdot Y \cdot pk_i^c \pmod{p}. \tag{16}$$

From 3) and 4), it follows that $\dot{M} = \bar{M}$, $p\dot{k}_i = p\bar{k}_i = pk_i$, $p\dot{k}_1 = p\bar{k}_1 = pk_1$, $\dot{\omega} = \bar{\omega} = \omega$, $\dot{Y} = \bar{Y} = Y$, $\dot{r} = \bar{r}$, $\dot{V} = \bar{V}$, $\dot{c} = \bar{c} = c$, $g^{\bar{z}} \equiv \bar{V} \cdot (X^{\bar{r}} \cdot Y \cdot pk_i^c)^{\bar{h}} \pmod{p}$, and $g^{\dot{z}} \equiv \bar{V} \cdot (X^{\bar{r}} \cdot Y \cdot pk_i^c)^{\dot{h}} \pmod{p}$. Since $\dot{h} \neq \bar{h}$, $(\bar{h} - \dot{h})^{-1}$ is well-defined. Thus,

$$g^{(\bar{z}-\dot{z})(\bar{h}-\dot{h})^{-1}} \equiv X^{\bar{r}} \cdot Y \cdot pk_i^c \pmod{p}. \tag{17}$$

Dividing Equation (16) by Equation (17) and raising both sides of the resulting congruence to the power $(r - \bar{r})^{-1}$ (which is well-defined since $\bar{r} \neq r$), we have

$$g^{\left((z-\hat{z})(h-\hat{h})^{-1}-(\bar{z}-\dot{z})(\bar{h}-\dot{h})^{-1}\right)\cdot(r-\bar{r})^{-1}} \equiv X \pmod{p},$$

as desired.

Let *frk* be defined as in Lemma C.5. Applying this lemma, we have

$$\begin{aligned}
&\Pr[\,E_2\,] \\
&\leq \quad acc + \frac{q_d(q_d-1+q_G)+q_{sd}(q_{sd}-1+q_G)+q_s(q_s-1+q_G)+q_p(q_p-1+q_H)+3}{q} \\
&\leq \quad \sqrt[4]{\beta^6 \cdot \mathit{frk}} \;+\; \sqrt[4]{\frac{3\cdot\beta^6}{q}} \;+ \\
&\qquad \frac{q_d(q_d-1+q_G)+q_{sd}(q_{sd}-1+q_G)+q_s(q_s-1+q_G)+q_p(q_p-1+q_H)+3}{q} \\
&\leq \quad \sqrt[4]{(q_R+q_H)^6 \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{C}}(\kappa)} \;+\; \sqrt[4]{\frac{3\cdot(q_R+q_H)^6}{q}} \;+ \\
&\qquad \frac{q_d(q_d-1+q_G)+q_{sd}(q_{sd}-1+q_G)+q_s(q_s-1+q_G)+q_p(q_p-1+q_H)+3}{q}\,.
\end{aligned}$$

This proves Equation (12).

Let $\gamma = q_G + q_H$ and $S = \mathbb{Z}_q$. Next, we define an algorithm $\boldsymbol{U}$ that given inputs a public key (p, q, g, X) and $s_1, \ldots, s_\gamma \in S$, returns a triple (I, J, σ) consisting of two integers $0 \leq J < I \leq \gamma$ and a string σ. Then we use the multiple-forking algorithm $\boldsymbol{MF}_{\boldsymbol{U},5}$ associated to $\boldsymbol{U}$ and 5 to construct adversary $\boldsymbol{D}$ against $\mathcal{G}_{\mathrm{dl}}$.

$\boldsymbol{U}$ makes the same initializations as algorithms $\boldsymbol{Y}$ and $\boldsymbol{Z}$ defined above: $n = 1$, $pk_1 = X$, and $j = 0$; it creates empty sets $\bar{S}$, S_1, $\mathbf{DU}$, and $\mathbf{CS}$; it creates empty arrays $\mathbf{skp}_1$, GT, RT, and HT; it chooses some randomness for $\boldsymbol{A}$; and then it runs $\boldsymbol{A}$ on input pk_1 with this randomness. $\boldsymbol{U}$ answers the requests and queries made by $\boldsymbol{A}$ exactly as $\boldsymbol{Y}$ does except for the differences specified below. These are identified by the number(s) of the corresponding step(s) in $\boldsymbol{Y}$.

1. If $\boldsymbol{A}$ makes a query $1||M$ to random oracle G, then $\boldsymbol{U}$ checks if $GT[1||M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[1||M] = c$. Then it returns $GT[1||M]$ to $\boldsymbol{A}$.

2. If $\boldsymbol{A}$ makes a query $0||pk_1||1||pk'_1||\omega||Y$ to random oracle G, then $\boldsymbol{U}$ checks if $GT[0||pk_1||1||pk'_1||\omega||Y]$ is defined. If not, it increments j and sets $0||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j = 0||pk_1||1||pk'_1||\omega||Y$ and $GT[0||pk_1||1||pk'_1||\omega||Y] = s_j$. Then it returns $GT[0||pk_1||1||pk'_1||\omega||Y]$ to $\boldsymbol{A}$.

2. If $\boldsymbol{A}$ makes a query $0||M$ that cannot be parsed as $0||pk_1||1||pk'_1||\omega||Y$ to random oracle G, then $\boldsymbol{U}$ checks if $GT[0||M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[0||M] = c$. Then it returns $GT[0||M]$ to $\boldsymbol{A}$.

3. If $\boldsymbol{A}$ makes a query M to random oracle R, then $\boldsymbol{U}$ checks if $RT[M]$ is defined. If not, it picks a random $r \in \mathbb{Z}_q$ and sets $RT[M] = r$. Then it returns $RT[M]$ to $\boldsymbol{A}$.

4. If $\boldsymbol{A}$ makes a query $0||M||pk_1||1||pk'_1||\omega||Y||r||V$ to random oracle H, then $\boldsymbol{U}$ checks if $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V]$ is defined. If not, it increments j and sets $0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||1||pk'_1||\omega||Y||r||V$ and $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V] = s_j$. Then it returns $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V]$ to $\boldsymbol{A}$.

4. If $\boldsymbol{A}$ makes a query M' that cannot be parsed as $0||M||pk_1||1||pk_1'||\omega||Y||r||V$ to random oracle H, then $\boldsymbol{U}$ checks if $HT[M']$ is defined. If not, it picks a random $h \in \mathbb{Z}_q$ and sets $HT[M'] = h$. Then it returns $HT[M']$ to $\boldsymbol{A}$.

Until $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. If $\boldsymbol{A}$'s forgery is not of the form $(M, p\sigma, pk_1)$, where $\mathcal{ID}(p\sigma) = 1$, then $\boldsymbol{U}$ aborts. Otherwise, $\boldsymbol{U}$ performs the following operations:

- Parse $p\sigma$ as $(1, \omega, Y, pk_1', (V, z))$
- If $\boldsymbol{A}$ did not make the following queries in the order given, then set $\mathsf{bad} = \mathsf{true}$.
  - $0||pk_1||1||pk_1'||\omega||Y$ to random oracle G,
  - $pk_1||1||pk_1'||\omega||Y||c$, where c is the response to the G-query above, to random oracle R,
  - $0||M||pk_1||1||pk_1'||\omega||Y||r||V$, where r is the response to the R-query above, to H

  Otherwise, set $c = GT[0||pk_1||1||pk_1'||\omega||Y]$, $r = RT[pk_1||1||pk_1'||\omega||Y||c]$, and $h = HT[0||M||pk_1|| 1||pk_1'||\omega||Y||r||V]$.
- If $\mathsf{bad} \neq \mathsf{true}$, compute proxy public key $pkp = {pk_1'}^r \cdot Y \cdot pk_1^c \bmod p$
- If $\mathsf{bad} = \mathsf{true}$ or $g^z \not\equiv V \cdot pkp^h \pmod p$ or $M \in S_1$ or $M \in \mathbf{CS}$, then return $(0, 0, \varepsilon)$
- Let j be such that $0||M_j||pk_{1,j}||1||pk_{1,j}'||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||1||pk_1'||\omega||Y||r||V$, and k such that $0||pk_{1,k}||1||pk_{1,k}'||\omega_k||Y_k = 0||pk_1||1||pk_1'||\omega||Y$
- Return $(j, k, (z, s_j, r, s_k))$

Let IG be the algorithm that runs $\mathcal{K}_{\mathsf{T}}(1^\kappa)$ to obtain (pk, sk) and returns $pk = (p, q, g, X)$. Let

$$acc \;=\; \Pr\left[\, pk \overset{\$}{\leftarrow} \mathsf{IG}\,;\; s_1, \ldots, s_\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q\,;\; (I, J, \sigma) \overset{\$}{\leftarrow} \boldsymbol{U}(pk, s_1, \ldots, s_\gamma) \;:\; I \geq 1 \wedge J \geq 1 \,\right],$$

as in Lemma C.5. Assume that event E_3 occurs and $\mathsf{bad} \neq \mathsf{true}$. Then when $\boldsymbol{A}$ makes query $0||pk_1||1||pk_1'||\omega||Y$ to random oracle G, $GT[0||pk_1||1||pk_1'||\omega||Y]$ is undefined and gets set to s_k for some k such that $1 \leq k \leq \gamma$. In addition, when $\boldsymbol{A}$ makes query $0||M||pk_1||1||pk_1'||\omega||Y||r||V$ to random oracle H, $HT[0||M||pk_1||1||pk_1'||\omega||Y||r||V]$ is undefined and gets set to s_j for some $j > k$ such that $1 \leq j \leq \gamma$. Therefore, $\boldsymbol{U}$ returns $(j, k, (z, s_j, r, s_k))$ for some $j > k \geq 1$. Thus,

$$\begin{aligned} acc &\geq \Pr[\, E_3 \wedge \mathsf{bad} \neq \mathsf{true} \,] \\ &\geq \Pr[\, E_3 \,] - \Pr[\, \mathsf{bad} = \mathsf{true} \mid E_3 \,] \\ &\geq \Pr[\, E_3 \,] - \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{|\mathbb{Z}_q|} \end{aligned}$$

Let $\boldsymbol{MF}_{\boldsymbol{U},5}$ be the multiple-forking algorithm associated to $\boldsymbol{U}$ as per Lemma C.5. Then we define adversary $\boldsymbol{D}$ against discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ as follows.

```
Adversary $\boldsymbol{D}(p, q, g, X)$
    $pk \leftarrow (p, q, g, X)$ ; $(b, \mathsf{results}) \overset{\$}{\leftarrow} \boldsymbol{MF}_{\boldsymbol{U},5}(pk)$
    If ( $b = 0$ ) then return 0
    Parse results[0] as $(z, h, r, c)$, results[1] as $(\hat{z}, \hat{h}, \hat{r}, \hat{c})$, results[2] as $(\bar{z}, \bar{h}, \bar{r}, \bar{c})$,
            results[3] as $(\dot{z}, \dot{h}, \dot{r}, \dot{c})$, results[4] as $(\check{z}, \check{h}, \check{r}, \check{c})$, results[5] as $(\tilde{z}, \tilde{h}, \tilde{r}, \tilde{c})$
    If ( $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \equiv 0 \pmod q$ ) then return 0
    else
        Solve the following system of equations modulo $q$ to obtain $x_3$:
            $r \cdot x_1 + x_2 + c \cdot x_3 \equiv (z - \hat{z})(h - \hat{h})^{-1}$
            $\bar{r} \cdot x_1 + x_2 + \bar{c} \cdot x_3 \equiv (\bar{z} - \dot{z})(\bar{h} - \dot{h})^{-1}$
            $\check{r} \cdot x_1 + x_2 + \check{c} \cdot x_3 \equiv (\check{z} - \tilde{z})(\check{h} - \tilde{h})^{-1}$
        return $x_3$
```

We claim that if $b = 1$ and $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod q$, then $\boldsymbol{D}$ computes the discrete logarithm of X. To justify this claim, consider the definitions of $\boldsymbol{U}$ and $\boldsymbol{MF}_{\boldsymbol{U},5}$. If $b = 1$ then there exist coins ρ for $\boldsymbol{U}$, $j \geq 1$, $k \geq 1$ and $s_1, \ldots, s_\gamma, s'_j, \ldots, s'_\gamma, t_k, \ldots, t_\gamma, t'_j, \ldots, t'_\gamma, u_k, \ldots, u_\gamma, u'_j, \ldots, u'_\gamma \in \mathbb{Z}_q$ with $\hat{h} = s'_j \neq s_j = h$, $\hat{c} = s_k = c$, $\bar{c} = t_k \neq s_k = \hat{c}$, $\dot{h} = t'_j \neq t_j = \bar{h}$, $\dot{c} = t_k = \bar{c}$, $\check{c} = u_k \neq t_k = \dot{c}$, $\tilde{h} = u'_j \neq u_j = \check{h}$, and $\tilde{c} = u_k = \check{c}$, such that

1) in the execution of $\boldsymbol{U}(pk), s_1, \ldots, s_\gamma; \rho)$, adversary $\boldsymbol{A}$ outputs a valid forgery $(M, (1, \omega, Y, pk'_1, (V, z)), pk_1)$ with $c = GT[0||pk_1||1||pk'_1||\omega||Y] = s_k$, $r = RT[pk_1||1||pk'_1||\omega||Y||c]$, $h = HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V] = s_j$, $0||pk_1||1||pk'_1||\omega||Y = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||M||pk_1||1||pk'_1||\omega||Y||r||V = 0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j$,

2) in the execution of $\boldsymbol{U}(pk, s_1, \ldots, s_{j-1}, s'_j, \ldots, s'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\hat{M}, (1, \hat{\omega}, \hat{Y}, \hat{pk}'_1, (\hat{V}, \hat{z})), \hat{pk}_1)$ with $\hat{c} = GT[0||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}] = s_k$, $\hat{r} = RT[\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{c}]$, $\hat{h} = HT[0||\hat{M}||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V}] = s'_j$, $0||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y} = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||\hat{M}||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V} = 0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j$,

3) in the execution of $\boldsymbol{U}(pk), s_1, \ldots, s_{k-1}, t_k, \ldots, t_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\bar{M}, (1, \bar{\omega}, \bar{Y}, \bar{pk}'_1, (\bar{V}, \bar{z})), \bar{pk}_1)$ with $\bar{c} = GT[0||\bar{pk}_1||1||\bar{pk}'_1||\bar{\omega}||\bar{Y}] = t_k$, $\bar{r} = RT[\bar{pk}_1||1||\bar{pk}'_1||\bar{\omega}||\bar{Y}||\bar{c}]$, $\bar{h} = HT[0||\bar{M}||\bar{pk}_1||1||\bar{pk}'_1||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V}] = t_j$, $0||\bar{pk}_1||1||\bar{pk}'_1||\bar{\omega}||\bar{Y} = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||\bar{M}||\bar{pk}_1||1||\bar{pk}'_1||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V} = 0||\bar{M}_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$,

4) in the execution of $\boldsymbol{U}(pk, s_1, \ldots, s_{k-1}, t_k, \ldots, t_{j-1}, t'_j, \ldots, t'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\dot{M}, (1, \dot{\omega}, \dot{Y}, \dot{pk}'_1, (\dot{V}, \dot{z})), \dot{pk}_1)$ with $\dot{c} = GT[0||\dot{pk}_1||1||\dot{pk}'_1||\dot{\omega}||\dot{Y}] = t_k$, $\dot{r} = RT[\dot{pk}_1||1||\dot{pk}'_1||\dot{\omega}||\dot{Y}||\dot{c}]$, $\dot{h} = HT[0||\dot{M}||\dot{pk}_1||1||\dot{pk}'_1||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V}] = t'_j$, $0||\dot{pk}_1||1||\dot{pk}'_1||\dot{\omega}||\dot{Y} = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||\dot{M}||\dot{pk}_1||1||\dot{pk}'_1||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V} = 0||\bar{M}_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$,

5) in the execution of $\boldsymbol{U}(pk, s_1, \ldots, s_{k-1}, u_k, \ldots, u_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\check{M}, (1, \check{\omega}, \check{Y}, \check{pk}'_1, (\check{V}, \check{z})), \check{pk}_1)$ with $\check{c} = GT[0||\check{pk}_1||1||\check{pk}'_1||\check{\omega}||\check{Y}] = u_k$, $\check{r} = RT[\check{pk}_1||1||\check{pk}'_1||\check{\omega}||\check{Y}||\check{c}]$, $\check{h} = HT[0||\check{M}||\check{pk}_1||1||\check{pk}'_1||\check{\omega}||\check{Y}||\check{r}||\check{V}] = u_j$, $0||\check{pk}_1||1||\check{pk}'_1||\check{\omega}||\check{Y} = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||\check{M}||\check{pk}_1||1||\check{pk}'_1||\check{\omega}||\check{Y}||\check{r}||\check{V} = 0||\check{M}_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||\check{r}_j||\check{V}_j$, and

6) in the execution of $\boldsymbol{U}(pk, s_1, \ldots, s_{k-1}, u_k, \ldots, u_{j-1}, u'_j, \ldots, u'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\tilde{M}, (1, \tilde{\omega}, \tilde{Y}, \tilde{pk}'_1, (\tilde{V}, \tilde{z})), \tilde{pk}_1)$ with $\tilde{c} = GT[0||\tilde{pk}_1||1||\tilde{pk}'_1||\tilde{\omega}||\tilde{Y}] = u_k$, $\tilde{r} = RT[\tilde{pk}_1||1||\tilde{pk}'_1||\tilde{\omega}||\tilde{Y}||\tilde{c}]$, $\tilde{h} = HT[0||\tilde{M}||\tilde{pk}_1||1||\tilde{pk}'_1||\tilde{\omega}||\tilde{Y}||\tilde{r}||\tilde{V}] = u'_j$, $0||\tilde{pk}_1||1||\tilde{pk}'_1||\tilde{\omega}||\tilde{Y} = 0||pk_{1,k}||1||pk'_{1,k}||\omega_k||Y_k$, and $0||\tilde{M}||\tilde{pk}_1||1||\tilde{pk}'_1||\tilde{\omega}||\tilde{Y}||\tilde{r}||\tilde{V} = 0||\check{M}_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||\check{r}_j||\check{V}_j$.

From 1) and 2), it follows that $\hat{M} = M$, $\hat{pk}_1 = pk_1$, $\hat{pk}'_1 = pk'_1$, $\hat{\omega} = \omega$, $\hat{Y} = Y$, $\hat{r} = r$, $\hat{V} = V$, $\hat{c} = c$, $g^z \equiv V \cdot ({pk'_1}^r \cdot Y \cdot X^c)^h \pmod p$, and $g^{\hat{z}} \equiv V \cdot ({pk'_1}^r \cdot Y \cdot X^c)^{\hat{h}} \pmod p$. Since $\hat{h} \neq h$, $(h - \hat{h})^{-1}$ exists. Thus,

$$g^{(z-\hat{z})(h-\hat{h})^{-1}} \equiv {pk'_1}^r \cdot Y \cdot X^c \pmod p. \tag{18}$$

From 3) and 4), it follows that $\dot{M} = \bar{M}$, $\dot{pk}_1 = \bar{pk}_1 = pk_1$, $\dot{pk}'_1 = \bar{pk}'_1 = pk'_1$, $\dot{\omega} = \bar{\omega} = \omega$, $\dot{Y} = \bar{Y} = Y$, $\dot{r} = \bar{r}$, $\dot{V} = \bar{V}$, $\dot{c} = \bar{c}$, $g^{\bar{z}} \equiv \bar{V} \cdot ({pk'_1}^{\bar{r}} \cdot Y \cdot X^{\bar{c}})^{\bar{h}} \pmod p$, and $g^{\dot{z}} \equiv \bar{V} \cdot ({pk'_1}^{\bar{r}} \cdot Y \cdot X^{\bar{c}})^{\dot{h}} \pmod p$. Since $\dot{h} \neq \bar{h}$, $(\bar{h} - \dot{h})^{-1}$ exists. Thus,

$$g^{(\bar{z}-\dot{z})(\bar{h}-\dot{h})^{-1}} \equiv {pk'_1}^{\bar{r}} \cdot Y \cdot X^{\bar{c}} \pmod p. \tag{19}$$

From 5) and 6), it follows that $\tilde{M} = \check{M}$, $\tilde{pk}_1 = \check{pk}_1 = pk_1$, $\tilde{pk}'_1 = \check{pk}'_1 = pk'_1$, $\tilde{\omega} = \check{\omega} = \omega$, $\tilde{Y} = \check{Y} = Y$, $\tilde{r} = \check{r}$, $\tilde{V} = \check{V}$, $\tilde{c} = \check{c}$, $g^{\check{z}} \equiv \check{V} \cdot ({pk'_1}^{\check{r}} \cdot Y \cdot X^{\check{c}})^{\check{h}} \pmod p$, and $g^{\tilde{z}} \equiv \check{V} \cdot ({pk'_1}^{\check{r}} \cdot Y \cdot X^{\check{c}})^{\tilde{h}} \pmod p$. Since

$\tilde{h} \neq \check{h}$, $(\check{h} - \tilde{h})^{-1}$ exists. Thus,

$$g^{(\check{z}-\tilde{z})(\check{h}-\tilde{h})^{-1}} \equiv {pk'_1}^{\check{r}} \cdot Y \cdot X^{\check{c}} \pmod{p}. \tag{20}$$

Equations (18), (19) and (20) yield the system of equations solved by $\boldsymbol{D}$, where $g^{x_1} = pk'_1$, $g^{x_2} = Y$ and $g^{x_3} = X$. If $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod{q}$, then the system has a unique solution and $\boldsymbol{D}$ returns the discrete logarithm of X.

Let *frk* be defined as in Lemma C.5. Then,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{D}}(k) &\geq \Pr\left[\, b = 1 \wedge r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod{q} \,\right] \\
&\geq \mathit{frk} - \Pr\left[\, r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \equiv 0 \pmod{q} \,\right] \\
&\geq \mathit{frk} - \frac{1}{q}\ .
\end{aligned}$$

The last equation above follows from the fact that values r, $\hat{r}$, $\bar{r}$, c, $\hat{c}$, $\bar{c}$ are independent and uniformly distributed, according to the definitions of $\boldsymbol{U}$ and $\boldsymbol{MF}_{\boldsymbol{U},5}$.

Applying Lemma C.5, we then have

$$\begin{aligned}
&\Pr\left[\, E_3 \,\right] \\
&\leq \; acc + \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{\gamma^{10} \cdot \mathit{frk}} \; + \; \sqrt[6]{\frac{5 \cdot \gamma^{10}}{q}} \; + \\
&\qquad \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{(q_G + q_H)^{10}\left(\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{D}}(\kappa) + 1/q\right)} \; + \; \sqrt[6]{\frac{5 \cdot (q_G + q_H)^{10}}{q}} \; + \\
&\qquad \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{(q_G + q_H)^{10} \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{D}}(\kappa)} \; + \; \sqrt[6]{\frac{(q_G + q_H)^{10}}{q}} \; + \; \sqrt[6]{\frac{5 \cdot (q_G + q_H)^{10}}{q}} \; + \\
&\qquad \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q}\ .
\end{aligned}$$

The last equation follows from the fact that $\sqrt[6]{a+b} \leq \sqrt[6]{a} + \sqrt[6]{b}$ for any real numbers $a, b \geq 0$. This proves Equation (13).

Let $\delta = q_H$ and $S = \mathbb{Z}_q$. We define an algorithm $\boldsymbol{V}$ that given inputs a public key (p, q, g, X) and $s_1, \ldots, s_\delta \in S$, returns a pair (I, σ) consisting of an integer $0 \leq I \leq \delta$ and a string σ. Then we use the forking algorithm $\boldsymbol{F}_{\boldsymbol{Y}}$ associated to $\boldsymbol{Y}$ to construct adversary $\boldsymbol{E}$ against $\mathcal{G}_{\mathrm{dl}}$.

$\boldsymbol{V}$ sets $n = 1$, $pk'_1 = X$, $j = 0$, and $\mathit{ctr} = 0$; creates empty sets $\bar{S}$, S_1, $\mathbf{DU}$, and $\mathbf{CS}$; creates empty arrays $\mathbf{skp}_1$, GT, RT, and HT; creates a key pair (pk_1, sk_1) by selecting $sk_1 \in \mathbb{Z}_q$ at random and setting $pk_1 = g^{sk_1} \bmod p$; chooses $m \in \{1, \ldots, q_{sd}\}$ at random; chooses some randomness for $\boldsymbol{A}$; and runs $\boldsymbol{A}$ on input pk_1 with this randomness. It then answers the requests and queries made by $\boldsymbol{A}$ as follows.

1. If $\boldsymbol{A}$ makes a query M to random oracle G, then $\boldsymbol{V}$ checks if $\mathit{GT}[M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $\mathit{GT}[M] = c$. Then it returns $\mathit{GT}[M]$ to $\boldsymbol{A}$.
2. If $\boldsymbol{A}$ makes a query M to random oracle R, then $\boldsymbol{V}$ checks if $\mathit{RT}[M]$ is defined. If not, it picks a random $r \in \mathbb{Z}_q$ and sets $\mathit{RT}[M] = r$. Then it returns $\mathit{RT}[M]$ to $\boldsymbol{A}$.

3. If $\boldsymbol{A}$ makes a query $0||M||pk_1||1||pk'_1||\omega||Y||r||V$ to random oracle H, then $\boldsymbol{V}$ checks if $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V]$ is defined. If not, it increments j and sets $0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||1||pk'_1||\omega||Y||r||V$ and $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V] = s_j$. Then it returns $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V]$ to $\boldsymbol{A}$.
4. If $\boldsymbol{A}$ makes a query M' that cannot be parsed as $0||M||pk_1||1||pk'_1||\omega||Y||r||V$ to random oracle H, then $\boldsymbol{V}$ checks if $HT[M']$ is defined. If not, it picks a random $h \in \mathbb{Z}_q$ and sets $HT[M'] = h$. Then it returns $HT[M']$ to $\boldsymbol{A}$.
5. If $\boldsymbol{A}$ requests to register a public key pk for user $n+1$, then $\boldsymbol{V}$ increments n, sets $pk_n = pk$, $S_n = \emptyset$ and creates an empty array $\mathbf{skp}_n$.
6. If $\boldsymbol{A}$ requests to interact with user 1 running $\mathcal{D}(pk_1, sk_1, i, pk_i, \omega)$, for some $i \in \{2, \ldots, n\}$, and play the role of user i running $\mathcal{P}(pk_i, sk_i, pk_1)$, then $\boldsymbol{V}$, sets $\mathbf{DU} = \mathbf{DU} \cup \{(i, \omega)\}$, and performs the following operations:
   - Pick a random $y \in \mathbb{Z}_q$
   - Set $Y = g^y \bmod p$
   - If $GT[0||pk_1||i||pk_i||\omega||Y]$ is defined, set $c = GT[0||pk_1||i||pk_i||\omega||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[0||pk_1||i||pk_i||\omega||Y] = c$.
   - Set $s = y + c \cdot sk_1 \bmod q$
   - Return $\omega, (Y, s)$ to $\boldsymbol{A}$

   Thus, $\boldsymbol{V}$ computes a Schnorr signature by user 1 for message $0||pk_1||i||pk_i||\omega$ using sk_1, and gives this signature and ω to $\boldsymbol{A}$.
7. If $\boldsymbol{A}$ requests to interact with user 1 running $\mathcal{P}(pk_1, sk_1, pk_i)$, for some $i \in \{2, \ldots, n\}$, and play the role of user i running $\mathcal{D}(pk_i, sk_i, 1, pk_1, \omega)$, when $\boldsymbol{A}$ outputs ω, (Y, s), $\boldsymbol{V}$ performs the following operations:
   - If $GT[0||pk_i||1||pk_1||\omega||Y]$ is defined, set $c = GT[0||pk_i||1||pk_1||\omega||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[0||pk_i||1||pk_1||\omega||Y] = c$.
   - Verify that (Y, s) is a valid signature for message $0||pk_i||1||pk_1||\omega$ with respect to public key pk_i (i.e., check that $g^s \equiv Y \cdot pk_i^c \pmod{p}$). If not, abort.
   - If $RT[pk_i||1||pk_1||\omega||Y||c]$ is defined, set $r = RT[pk_i||1||pk_1||\omega||Y||c]$. Otherwise, pick a random $r \in \mathbb{Z}_q$ and set $RT[pk_i||1||pk_1||\omega||Y||c] = r$.
   - Set $t = r \cdot sk_1 + s \bmod q$
   - Set $skp = (pk_i||1||pk_1||\omega, Y, t)$
   - Store (skp, ω) in the last unoccupied position of $\mathbf{skp}_i$

   Here $\boldsymbol{V}$ computes a correct proxy signing key for user 1 using sk_1.
8. If $\boldsymbol{A}$ requests that user 1 run the designation protocol with itself for ω, then $\boldsymbol{V}$ increments ctr. If $ctr \neq m$ then $\boldsymbol{V}$ creates a new key pair (pk''_1, sk''_1) by selecting $sk''_1 \in \mathbb{Z}_q$ at random and setting $pk''_1 = g^{sk''_1} \bmod p$, and performs the following operations:
   - Pick a random $y \in \mathbb{Z}_q$
   - Set $Y = g^y \bmod p$
   - If $GT[0||pk_1||1||pk''_1||\omega||Y]$ is defined, set $c = GT[0||pk_1||1||pk''_1||\omega||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[0||pk_1||1||pk''_1||\omega||Y] = c$.
   - Set $s = y + c \cdot sk_1 \bmod q$
   - If $RT[pk_1||1||pk''_1||\omega||Y||c]$ is defined, set $r = RT[pk_1||1||pk''_1||\omega||Y||c]$. Otherwise, pick a random $r \in \mathbb{Z}_q$ and set $RT[pk_1||1||pk''_1||\omega||Y||c] = r$.
   - Set $t = r \cdot sk''_1 + s \bmod q$

- Set $skp = (pk_1||1||pk_1''||\omega, Y, t)$
- Store (skp, ω) in $\mathbf{skp}_1[ctr]$
- Return $\omega, (Y, s)$ to $\boldsymbol{A}$

Here $\boldsymbol{V}$ computes a Schnorr signature by user 1 for message $0||pk_1||1||pk_1''||\omega$. Using the signature obtained, it computes a correct proxy signing key for user 1.

Otherwise (i.e., $ctr = m$), $\boldsymbol{V}$ creates an appropriate message space description ω, and performs the following operations:

- Pick a random $y \in \mathbb{Z}_q$
- Set $Y = g^y \bmod p$
- If $GT[0||pk_1||1||pk_1'||\omega||Y]$ is defined, set $c = GT[0||pk_1||1||pk_1'||\omega||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[0||pk_1||1||pk_1'||\omega||Y] = c$.
- Set $s = y + c \cdot sk_1 \bmod q$
- Store ω, Y, s in $\mathbf{skp}_1[ctr]$.
- Return $\omega, (Y, s)$ to $\boldsymbol{A}$

Here $\boldsymbol{V}$ computes a Schnorr signature by user 1 for message $0||pk_1||1||pk_1'||\omega$ using sk_1, and gives the certificate, which is the message space description and the signature, to $\boldsymbol{A}$.

9. If $\boldsymbol{A}$ requests to see $\mathbf{skp}_1[l]$ for some $l \in \mathbb{N}$, then if $\mathbf{skp}_1[l]$ contains a pair (skp, ω), $\boldsymbol{V}$ sets $\mathbf{CS} = \mathbf{CS} \cup \omega$ and returns skp to $\boldsymbol{A}$; otherwise, if $\mathbf{skp}_1[l]$ contains ω, an element $Y \in \mathbb{Z}_p$, and an element $s \in \mathbb{Z}_q$, $\boldsymbol{V}$ aborts. Otherwise, $\boldsymbol{V}$ returns $\perp$ to $\boldsymbol{A}$.

10. If $\boldsymbol{A}$ queries its oracle $\mathcal{O}_{\mathcal{S}_\mathsf{T}}(sk_1, \cdot)$ with a message M, then $\boldsymbol{V}$ performs the following operations:
    - Pick a random $y \in \mathbb{Z}_q$
    - Set $Y = g^y \bmod p$
    - If $GT[1||M||Y]$ is defined, set $c = GT[1||M||Y]$. Otherwise, pick a random $c \in \mathbb{Z}_q$ and set $GT[1||M||Y] = c$.
    - Set $s = y + c \cdot sk_1 \bmod q$
    - Set $\bar{S} = \bar{S} \cup \{M\}$
    - Return (Y, s) to $\boldsymbol{A}$

    $\boldsymbol{V}$ computes a Schnorr signature by user 1 for message $1||M$ using sk_1.

11. If $\boldsymbol{A}$ makes a query (i, l, M), where $i \in \{2, \ldots, n\}$, $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, then $\boldsymbol{V}$ responds as follows. If $\mathbf{skp}_i[l]$ is not defined, then it returns $\perp$ to $\boldsymbol{A}$. Otherwise, it parses $\mathbf{skp}_i[l]$ as $((pk_i||1||pk_1||\omega_l, Y_l, t_l), \omega_l)$, and performs the following operations:
    - Pick a random $y \in \mathbb{Z}_q$
    - Compute commitment $Y \leftarrow g^y \bmod p$
    - Set $c = GT[0||pk_i||1||pk_1||\omega_l||Y_l]$
    - Set $r = RT[pk_i||1||pk_1||\omega_l||Y_l||c]$
    - If $HT[0||M||pk_i||1||pk_1||\omega_l||Y_l||r||Y]$ is defined, set $h = HT[0||M||pk_i||1||pk_1||\omega_l||Y_l||r||Y]$. Otherwise, pick a random $h \in \mathbb{Z}_q$ and set $HT[0||M||pk_i||1||pk_1||\omega_l||Y_l||r||Y] = h$.
    - Set $s = y + t_l \cdot h \bmod q$
    - Set $S_i = S_i \cup \{M\}$
    - Return $(1, \omega_l, Y_l, pk_1, (Y, s))$ to $\boldsymbol{A}$

    Thus, $\boldsymbol{V}$ computes a proxy signature by user 1 on behalf of user i for message $0||M||pk_i||1||pk_1||\omega_l||Y_l||r$, using the l-th proxy signing key and the signature returned to adversary $\boldsymbol{A}$ is identical to the signature returned by oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$.

12. If $\boldsymbol{A}$ makes a query $(1, l, M)$, where $l \in \mathbb{N}$, and $M \in \{0,1\}^*$, to its oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$, then $\boldsymbol{V}$ responds as follows. If $\mathbf{skp}_1[l]$ is not defined, then it returns $\perp$ to $\boldsymbol{A}$. Otherwise, if $\mathbf{skp}_1[l]$ contains a pair (skp, ω), $\boldsymbol{V}$ parses skp as $(pk_1||1||pk_1^l||\omega_l, Y_l, t_l)$, and performs the following operations:
    - Pick a random $y \in \mathbb{Z}_q$
    - Compute commitment $Y \leftarrow g^y \bmod p$
    - Set $c = GT[0||pk_1||1||pk_1^l||\omega_l||Y_l]$
    - Set $r = RT[pk_1||1||pk_1^l||\omega_l||Y_l||c]$
    - If $HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y]$ is defined, set $h = HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y]$. Otherwise, pick a random $h \in \mathbb{Z}_q$ and set $HT[0||M||pk_1||1||pk_1^l||\omega_l||Y_l||r||Y] = h$.
    - Set $s = y + t_l \cdot h \bmod q$
    - Set $S_1 = S_1 \cup \{M\}$
    - Return $(1, \omega_l, Y_l, pk_1^l, (Y, s))$ to $\boldsymbol{A}$

    In this case, $\boldsymbol{V}$ computes a proxy signature by user 1 on behalf of herself using the l-th proxy signing key $(pk_1||1||pk_1^l||\omega_l, Y_l, t_l)$. The signature returned to adversary $\boldsymbol{A}$ is thus identical to the signature returned by oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$.

    Otherwise, if $\mathbf{skp}_1[l]$ contains ω_l, an element $Y_l \in \mathbb{Z}_p$, and an element $s_l \in \mathbb{Z}_q$, $\boldsymbol{V}$ performs the following operations:
    - Pick a random $h \in \mathbb{Z}_q$
    - Pick a random $s \in \mathbb{Z}_q$
    - Set $c = GT[0||pk_1||1||pk'_1||\omega_l||Y_l]$
    - If $RT[pk_1||1||pk'_1||\omega_l||Y_l||c]$ is defined, set $r = RT[pk_1||1||pk'_1||\omega_l||Y_l||c]$. Otherwise, pick a random $r \in \mathbb{Z}_q$ and set $RT[pk_1||1||pk'_1||\omega_l||Y_l||c] = r$.
    - Compute proxy public key $pkp = {pk'_1}^r \cdot Y_l \cdot pk_1^c \bmod p$
    - Compute commitment $Y \leftarrow g^s \cdot pkp^{-h} \bmod p$
    - If $HT[0||M||pk_1||1||pk'_1||\omega_l||Y_l||r||Y]$ is defined, set $\mathsf{bad} = \mathsf{true}$
    - Set $HT[0||M||pk_1||1||pk'_1||\omega_l||Y_l||r||Y] = h$
    - Set $S_1 = S_1 \cup \{M\}$
    - Return $(1, \omega_l, Y_l, pk'_1, (Y, s))$ to $\boldsymbol{A}$

    In this case, $\boldsymbol{V}$ simulates proxy signing by user 1 on behalf of herself using the l-th proxy signing key. It is easy to see that the simulated signature (Y, s) has the same distribution as a real Schnorr signature for message $0||M||pk_1||1||pk'_1||\omega_l||Y_l||r$. Therefore, the signature returned to adversary $\boldsymbol{A}$ has the same distribution as a signature returned by oracle $\mathcal{O}_{\mathcal{PS}}((\mathbf{skp}_u)_{u\in[n]}, \cdot, \cdot, \cdot)$.

Until $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. If $\boldsymbol{A}$'s forgery is not of the form $(M, p\sigma, pk_1)$, where $\mathcal{ID}(p\sigma) = 1$, then $\boldsymbol{V}$ aborts. Otherwise, $\boldsymbol{V}$ performs the following operations:

- Parse $p\sigma$ as $(1, \omega, Y, pk'_1, (V, z))$
- If $\boldsymbol{A}$ did not make the following queries in the order given, then set $\mathsf{bad} = \mathsf{true}$.
    - $0||pk_1||1||pk'_1||\omega||Y$ to random oracle G,
    - $pk_1||1||pk'_1||\omega||Y||c$, where c is the response to the G-query above, to random oracle R,
    - $0||M||pk_1||1||pk'_1||\omega||Y||r||V$, where r is the response to the R-query above, to H

  Otherwise, set $c = GT[0||pk_1||1||pk'_1||\omega||Y]$, $r = RT[pk_1||1||pk'_1||\omega||Y||c]$, and $h = HT[0||M||pk_1|| 1||pk'_1||\omega||Y||r||V]$.
- If $\mathsf{bad} \neq \mathsf{true}$, compute proxy public key $pkp = {pk'_1}^r \cdot Y \cdot pk_1^c \bmod p$
- If $\mathsf{bad} = \mathsf{true}$ or $g^z \not\equiv V \cdot pkp^h \pmod p$ or $M \in S_1$ or $M \in \mathbf{CS}$, then return $(0, \varepsilon)$

- If $\mathbf{skp}_1[m] \neq \omega, Y, s$ for some $s \in \mathbb{Z}_q$, then return $(0, \varepsilon)$
- Let j be such that $0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||1||pk'_1||\omega||Y||r||V$
- Return $(j, (z, r, s_j, s))$

Let IG be the algorithm that runs $\mathcal{K}_{\mathsf{T}}(1^\kappa)$ to obtain (pk, sk) and returns $pk = (p, q, g, X)$. Let

$$acc \;=\; \Pr\Big[\, pk \xleftarrow{\$} \mathsf{IG}\;;\; s_1, \ldots, s_\delta \xleftarrow{\$} \mathbb{Z}_q\;;\; (I, \sigma) \xleftarrow{\$} \boldsymbol{V}(pk, s_1, \ldots, s_\delta) \;:\; I \geq 1 \,\Big],$$

as in Lemma C.1. Assume that event E_4 occurs, $\mathsf{bad} \neq \mathsf{true}$ and $\boldsymbol{V}$ correctly guesses which self-delegation request was answered with a Schnorr signature (Y, s) for message $0||pk_1||1||pk'_1||\omega$. Then $\boldsymbol{A}$ does not request to see $skp_1[m]$, so $\boldsymbol{V}$ does not abort in step 9 above. Additionally, $skp_1[m] = \omega, Y, s$. When $\boldsymbol{A}$ makes query $0||M||pk_1||1||pk'_1||\omega||Y||r||V$ to random oracle H, $HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V]$ is undefined and gets set to s_i for some i such that $1 \leq i \leq \delta$. Therefore, $\boldsymbol{V}$ returns $(i, (z, r, s_i, s))$ for some $i \geq 1$. Thus,

$$\begin{aligned}
acc &\geq \Pr[\, E_4 \wedge \mathsf{bad} \neq \mathsf{true} \wedge \boldsymbol{V} \text{ guesses correctly} \,] \\
&= \Pr[\, E_4 \wedge \mathsf{bad} \neq \mathsf{true} \,] \cdot \frac{1}{q_{sd}} \\
q_{sd} \cdot acc &\geq \Pr[\, E_4 \,] - \Pr[\, \mathsf{bad} = \mathsf{true} \mid E_4 \,] \\
&\geq \Pr[\, E_4 \,] - \frac{3}{|\mathbb{Z}_q|}
\end{aligned}$$

Let $\boldsymbol{F_V}$ be the forking algorithm associated to $\boldsymbol{V}$ as per Lemma C.1. Then we define adversary $\boldsymbol{E}$ against discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ as follows.

```
Adversary $\boldsymbol{E}(p, q, g, X)$
    $pk \leftarrow (p, q, g, X)$ ; $(b, \sigma, \hat{\sigma}) \xleftarrow{\$} \boldsymbol{F_V}(pk)$
    If ( $b = 0$ ) then return 0
    Parse $\sigma$ as $(z, r, h, s)$ and $\hat{\sigma}$ as $(\hat{z}, \hat{r}, \hat{h}, \hat{s})$
    If ( $r \equiv 0 \pmod{q}$ ) then return 0
    else return $((z - \hat{z})(h - \hat{h})^{-1} - s) \cdot r^{-1} \bmod q$
```

We claim that if $b = 1$ and $r \not\equiv 0 \pmod{q}$, then $\boldsymbol{E}$ computes the discrete logarithm of X. To justify this claim, consider the definitions of $\boldsymbol{V}$ and $\boldsymbol{F_V}$. If $b = 1$ then there exist coins ρ for $\boldsymbol{V}$, $j \geq 1$ and $s_1, \ldots, s_\delta, s'_j, \ldots, s'_\delta \in \mathbb{Z}_q$ with $\hat{h} = s'_j \neq s_j = h$ such that

1) in the execution of $\boldsymbol{V}(pk, s_1, \ldots, s_\delta; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(M, (1, \omega, Y, pk'_1, (V, z)), pk_1)$ with $c = GT[0||pk_1||1||pk'_1||\omega||Y]$, $r = RT[pk_1||1||pk'_1||\omega||Y||c]$, $h = HT[0||M||pk_1||1||pk'_1||\omega||Y||r||V] = s_j$, $0||M||pk_1||1||pk'_1||\omega||Y||r||V = 0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j$, and $\mathbf{skp}_1[m] = \omega, Y, s$, where $g^s = Y \cdot pk_1^c \bmod p$, and

2) in the execution of $\boldsymbol{V}(pk, s_1, \ldots, s_{j-1}, s'_j, \ldots, s'_\delta; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\hat{M}, (1, \hat{\omega}, \hat{Y}, \hat{pk}'_1, (\hat{V}, \hat{z})), \hat{pk}_1)$ with $\hat{c} = GT[0||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}]$, $\hat{r} = RT[\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{c}]$, $\hat{h} = HT[0||\hat{M}||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V}] = s'_j$, $0||\hat{M}||\hat{pk}_1||1||\hat{pk}'_1||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V} = 0||M_j||pk_{1,j}||1||pk'_{1,j}||\omega_j||Y_j||r_j||V_j$, and $\mathbf{skp}_1[m] = \hat{\omega}, \hat{Y}, \hat{s}$, where $g^{\hat{s}} = \hat{Y} \cdot \hat{pk}_1^{\hat{c}} \bmod p$

It follows that $\hat{M} = M$, $\hat{pk}_1 = pk_1$, $\hat{pk}'_1 = pk'_1$, $\hat{\omega} = \omega$, $\hat{Y} = Y$, $\hat{r} = r$, $\hat{V} = V$, $\hat{c} = c$, $g^z \equiv V \cdot (X^r \cdot Y \cdot pk_1^c)^h \pmod{p}$, $g^{\hat{z}} \equiv V \cdot (X^r \cdot Y \cdot pk_1^c)^{\hat{h}} \pmod{p}$, $\hat{s} = s$, and $g^s = Y \cdot pk_1^c \bmod p$. Since $\hat{h} \neq h$, $(h - \hat{h})^{-1}$ exists. Thus,

$$g^{(z-\hat{z})(h-\hat{h})^{-1}} \equiv X^r \cdot g^s \pmod{p}.$$

If $r \not\equiv 0 \pmod{q}$, then we have

$$g^{((z-\hat{z})(h-\hat{h})^{-1}-s)\cdot r^{-1}} \equiv X \pmod{p}.$$

Therefore, $\boldsymbol{E}$ returns the discrete logarithm of X.

Let *frk* be defined as in Lemma C.1. Then,

$$\begin{array}{rcl} \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{E}}(k) & \geq & \Pr\left[\, b = 1 \wedge r \not\equiv 0 \pmod{q} \,\right] \\ & \geq & \mathit{frk} - \Pr\left[\, r \equiv 0 \pmod{q} \,\right] \\ & \geq & \mathit{frk} - \dfrac{1}{q}\ . \end{array}$$

The last equation above follows from the fact that r is uniformly distributed, according to the definitions of $\boldsymbol{V}$ and $\boldsymbol{F_V}$.

Applying Lemma C.1, we then have

$$\begin{array}{rcl} \Pr\left[\, E_4 \,\right] & \leq & q_{sd} \cdot acc + \dfrac{3}{q} \leq q_{sd} \cdot \left(\sqrt{\delta \cdot \mathit{frk}}\ + \dfrac{\delta}{q} \right) + \dfrac{3}{q} \\ & \leq & q_{sd} \cdot \sqrt{\delta \cdot \left(\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{E}}(\kappa) + 1/q \right)}\ + \dfrac{\delta \cdot q_{sd} + 3}{q} \\ & \leq & q_{sd} \cdot \sqrt{\delta \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{E}}(\kappa)} + q_{sd} \cdot \sqrt{\dfrac{\delta}{q}}\ + \dfrac{\delta \cdot q_{sd} + 3}{q}\ . \end{array}$$

The last equation follows from the fact that $\sqrt[6]{a+b} \leq \sqrt[6]{a} + \sqrt[6]{b}$ for any real numbers $a, b \geq 0$. This proves Equation (14).

Let $\gamma = q_G + q_H$ and $S = \mathbb{Z}_q$, as above. We now define an algorithm $\boldsymbol{W}$ that given inputs a public key (p, q, g, X) and $s_1, \ldots, s_\gamma \in S$, returns a triple (I, J, σ) consisting of two integers $0 \leq J < I \leq \gamma$ and a string σ. Then we use the multiple-forking algorithm $\boldsymbol{MF}_{\boldsymbol{W},5}$ associated to $\boldsymbol{W}$ and 5 to construct adversary $\boldsymbol{F}$ against $\mathcal{G}_{\mathrm{dl}}$.

$\boldsymbol{W}$ makes the same initializations as algorithm $\boldsymbol{Y}$ defined above: $n = 1$, $pk_1 = X$, and $j = 0$; it creates empty sets $\bar{S}$, S_1, $\mathbf{DU}$, and $\mathbf{CS}$; it creates empty arrays $\mathbf{skp}_1$, GT, RT, and HT; it chooses some randomness for $\boldsymbol{A}$; and then it runs $\boldsymbol{A}$ on input pk_1 with this randomness. $\boldsymbol{W}$ answers the requests and queries made by $\boldsymbol{A}$ exactly as $\boldsymbol{Y}$ does except for the differences specified below. These are identified by the number(s) of the corresponding step(s) in $\boldsymbol{Y}$.

1. If $\boldsymbol{A}$ makes a query $1||M$ to random oracle G, then $\boldsymbol{W}$ checks if $GT[1||M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[1||M] = c$. Then it returns $GT[1||M]$ to $\boldsymbol{A}$.
2. If $\boldsymbol{A}$ makes a query $0||pk_1||i||pk_i||\omega||Y$ to random oracle G, where $i \in \{2, \ldots, n\}$, then $\boldsymbol{W}$ checks if $GT[0||pk_1||i||pk_i||\omega||Y]$ is defined. If not, it increments j and sets $0||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j = 0||pk_1|| i||pk_i||\omega||Y$ and $GT[0||pk_1||i||pk_i||\omega||Y] = s_j$. Then it returns $GT[0||pk_1||i||pk_i||\omega||Y]$ to $\boldsymbol{A}$.
3. If $\boldsymbol{A}$ makes a query $0||M$ that cannot be parsed as $0||pk_1||i||pk_i||\omega||Y$, for some $i \in \{2, \ldots, n\}$, to random oracle G, then $\boldsymbol{W}$ checks if $GT[0||M]$ is defined. If not, it picks a random $c \in \mathbb{Z}_q$ and sets $GT[0||M] = c$. Then it returns $GT[0||M]$ to $\boldsymbol{A}$.
4. If $\boldsymbol{A}$ makes a query M to random oracle R, then $\boldsymbol{W}$ checks if $RT[M]$ is defined. If not, it picks a random $r \in \mathbb{Z}_q$ and sets $RT[M] = r$. Then it returns $RT[M]$ to $\boldsymbol{A}$.
5. If $\boldsymbol{A}$ makes a query $0||M||pk_1||i||pk_i||\omega||Y||r||V$ to random oracle H, where $i \in \{2, \ldots, n\}$, then $\boldsymbol{W}$ checks if $HT[0||M||pk_1||i||pk_i||\omega||Y||r||V]$ is defined. If not, it increments j and sets $0||M_j||pk_{1,j}|| i||pk_{i,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||i||pk_i||\omega||Y||r||V$ and $HT[0||M||pk_1||i||pk_i||\omega||Y||r||V] = s_j$. Then it returns $HT[0||M||pk_1||i||pk_i||\omega||Y||r||V]$ to $\boldsymbol{A}$.

6. If $\boldsymbol{A}$ makes a query M' that cannot be parsed as $0||M||pk_1||i||pk_i||\omega||Y||r||V$, for some $i \in \{2, \ldots, n\}$, to random oracle H, then $\boldsymbol{W}$ checks if $HT[M']$ is defined. If not, it picks a random $h \in \mathbb{Z}_q$ and sets $HT[M'] = h$. Then it returns $HT[M']$ to $\boldsymbol{A}$.

Until $\boldsymbol{A}$ outputs a forgery (M, σ) or $(M, p\sigma, pk)$. If $\boldsymbol{A}$'s forgery is not of the form $(M, p\sigma, pk_1)$, where $\mathcal{ID}(p\sigma) = i$ for some $i \in \{2, \ldots, n\}$, then $\boldsymbol{W}$ aborts. Otherwise, $\boldsymbol{W}$ performs the following operations:

- Parse $p\sigma$ as $(i, \omega, Y, pk_i, (V, z))$
- If $\boldsymbol{A}$ did not make the following queries in the order given, then set $\mathsf{bad} = \mathsf{true}$.
  - $0||pk_1||i||pk_i||\omega||Y$ to random oracle G,
  - $pk_1||i||pk_i||\omega||Y||c$, where c is the response to the G-query above, to random oracle R,
  - $0||M||pk_1||i||pk_i||\omega||Y||r||V$, where r is the response to the R-query above, to H

  Otherwise, set $c = GT[0||pk_1||i||pk_i||\omega||Y]$, $r = RT[pk_1||i||pk_i||\omega||Y||c]$, and $h = HT[0||M||pk_1||i||pk_i||\omega||Y||r||V]$.
- If $\mathsf{bad} \neq \mathsf{true}$, compute proxy public key $pkp = pk_i{}^r \cdot Y \cdot pk_1^c \bmod p$
- If $\mathsf{bad} = \mathsf{true}$ or $g^z \not\equiv V \cdot pkp^h \pmod p$, then return $(0, 0, \varepsilon)$
- Let j be such that $0||M_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||r_j||V_j = 0||M||pk_1||i||pk_i||\omega||Y||r||V$, and k such that $0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k = 0||pk_1||i||pk_i||\omega||Y$
- Return $(j, k, (z, s_j, r, s_k))$

Let IG be the algorithm that runs $\mathcal{K}_{\mathsf{T}}(1^\kappa)$ to obtain (pk, sk) and returns $pk = (p, q, g, X)$. Let

$$acc \ = \ \Pr\left[pk \xleftarrow{\$} \mathsf{IG} \,;\, s_1, \ldots, s_\gamma \xleftarrow{\$} \mathbb{Z}_q \,;\, (I, J, \sigma) \xleftarrow{\$} \boldsymbol{W}(pk, s_1, \ldots, s_\gamma) \ : \ I \geq 1 \wedge J \geq 1 \right],$$

as in Lemma C.5. Assume that event E_5 occurs and $\mathsf{bad} \neq \mathsf{true}$. Then when $\boldsymbol{A}$ makes query $0||pk_1||i||pk_i||\omega||Y$ to random oracle G, $GT[0||pk_1||i||pk_i||\omega||Y]$ is undefined and gets set to s_k for some k such that $1 \leq k \leq \gamma$. In addition, when $\boldsymbol{A}$ makes query $0||M||pk_1||i||pk_i||\omega||Y||r||V$ to random oracle H, $HT[0||M||pk_1||i||pk_i||\omega||Y||r||V]$ is undefined and gets set to s_j for some $j > k$ such that $1 \leq j \leq \gamma$. Therefore, $\boldsymbol{W}$ returns $(j, k, (z, s_j, r, s_k))$ for some $j > k \geq 1$. Thus,

$$\begin{aligned} acc &\geq \Pr[\, E_5 \wedge \mathsf{bad} \neq \mathsf{true} \,] \\ &\geq \Pr[\, E_5 \,] - \Pr[\, \mathsf{bad} = \mathsf{true} \mid E_5 \,] \\ &\geq \Pr[\, E_5 \,] - \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{|\mathbb{Z}_q|} \end{aligned}$$

Let $\boldsymbol{MF}_{\boldsymbol{W},5}$ be the multiple-forking algorithm associated to $\boldsymbol{W}$ as per Lemma C.5. Then we define adversary $\boldsymbol{F}$ against discrete-logarithm parameter generator $\mathcal{G}_{\mathrm{dl}}$ as follows.

```
Adversary $\boldsymbol{F}(p, q, g, X)$
    $pk \leftarrow (p, q, g, X)$ ; $(b, \mathsf{results}) \xleftarrow{\$} \boldsymbol{MF}_{\boldsymbol{W},5}(pk)$
    If ( $b = 0$ ) then return 0
    Parse results[0] as $(z, h, r, c)$, results[1] as $(\hat{z}, \hat{h}, \hat{r}, \hat{c})$, results[2] as $(\bar{z}, \bar{h}, \bar{r}, \bar{c})$,
            results[3] as $(\dot{z}, \dot{h}, \dot{r}, \dot{c})$, results[4] as $(\check{z}, \check{h}, \check{r}, \check{c})$, results[5] as $(\tilde{z}, \tilde{h}, \tilde{r}, \tilde{c})$
    If ( $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \equiv 0 \pmod q$ ) then return 0
    else
        Solve the following system of equations modulo $q$ to obtain $x_3$:
                $r \cdot x_1 + x_2 + c \cdot x_3 \equiv (z - \hat{z})(h - \hat{h})^{-1}$
                $\bar{r} \cdot x_1 + x_2 + \bar{c} \cdot x_3 \equiv (\bar{z} - \dot{z})(\bar{h} - \dot{h})^{-1}$
                $\check{r} \cdot x_1 + x_2 + \check{c} \cdot x_3 \equiv (\check{z} - \tilde{z})(\check{h} - \tilde{h})^{-1}$
        return $x_3$
```

We claim that if $b = 1$ and $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod{q}$, then $\boldsymbol{F}$ computes the discrete logarithm of X. To justify this claim, consider the definitions of $\boldsymbol{W}$ and $\boldsymbol{MF}_{\mathbf{W},5}$. If $b = 1$ then there exist coins ρ for $\boldsymbol{W}$, $j \geq 1$, $k \geq 1$ and $s_1, \ldots, s_\gamma, s'_j, \ldots, s'_\gamma, t_k, \ldots, t_\gamma, t'_j, \ldots, t'_\gamma, u_k, \ldots, u_\gamma, u'_j, \ldots, u'_\gamma \in \mathbb{Z}_q$ with $\hat{h} = s'_j \neq s_j = h$, $\hat{c} = s_k = c$, $\bar{c} = t_k \neq s_k = \hat{c}$, $\dot{h} = t'_j \neq t_j = \bar{h}$, $\dot{c} = t_k = \bar{c}$, $\check{c} = u_k \neq t_k = \dot{c}$, $\tilde{h} = u'_j \neq u_j = \check{h}$, and $\tilde{c} = u_k = \check{c}$, such that

1) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_\gamma; \rho)$, adversary $\boldsymbol{A}$ outputs a valid forgery $(M, (i, \omega, Y, pk_i, (V, z)), pk_1)$ with $c = GT[0||pk_1||i||pk_i||\omega||Y] = s_k$, $r = RT[pk_1||i||pk_i||\omega||Y||c]$, $h = HT[0||M||pk_1||i||pk_i||\omega||Y||r||V] = s_j$, $0||pk_1||i||pk_i||\omega||Y = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||M||pk_1||i||pk_i||\omega||Y||r||V = 0||M_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||r_j||V_j$,

2) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_{j-1}, s'_j, \ldots, s'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\hat{M}, (i, \hat{\omega}, \hat{Y}, p\hat{k}_i, (\hat{V}, \hat{z})), p\hat{k}_1)$ with $\hat{c} = GT[0||p\hat{k}_1||i||p\hat{k}_i||\hat{\omega}||\hat{Y}] = s_k$, $\hat{r} = RT[p\hat{k}_1||i||p\hat{k}_i||\hat{\omega}||\hat{Y}||\hat{c}]$, $\hat{h} = HT[0||\hat{M}||p\hat{k}_1||i||p\hat{k}_i||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V}] = s'_j$, $0||p\hat{k}_1||i||p\hat{k}_i||\hat{\omega}||\hat{Y} = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||\hat{M}||p\hat{k}_1||i||p\hat{k}_i||\hat{\omega}||\hat{Y}||\hat{r}||\hat{V} = 0||M_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||r_j||V_j$,

3) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_{k-1}, t_k, \ldots, t_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\bar{M}, (i, \bar{\omega}, \bar{Y}, p\bar{k}_i, (\bar{V}, \bar{z})), p\bar{k}_1)$ with $\bar{c} = GT[0||p\bar{k}_1||i||p\bar{k}_i||\bar{\omega}||\bar{Y}] = t_k$, $\bar{r} = RT[p\bar{k}_1||i||p\bar{k}_i||\bar{\omega}||\bar{Y}||\bar{c}]$, $\bar{h} = HT[0||\bar{M}||p\bar{k}_1||i||p\bar{k}_i||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V}] = t_j$, $0||p\bar{k}_1||i||p\bar{k}_i||\bar{\omega}||\bar{Y} = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||\bar{M}||p\bar{k}_1||i||p\bar{k}_i||\bar{\omega}||\bar{Y}||\bar{r}||\bar{V} = 0||\bar{M}_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$,

4) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_{k-1}, t_k, \ldots, t_{j-1}, t'_j, \ldots, t'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\dot{M}, (i, \dot{\omega}, \dot{Y}, p\dot{k}_i, (\dot{V}, \dot{z})), p\dot{k}_1)$ with $\dot{c} = GT[0||p\dot{k}_1||i||p\dot{k}_i||\dot{\omega}||\dot{Y}] = t_k$, $\dot{r} = RT[p\dot{k}_1||i||p\dot{k}_i||\dot{\omega}||\dot{Y}||\dot{c}]$, $\dot{h} = HT[0||\dot{M}||p\dot{k}_1||i||p\dot{k}_i||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V}] = t'_j$, $0||p\dot{k}_1||i||p\dot{k}_i||\dot{\omega}||\dot{Y} = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||\dot{M}||p\dot{k}_1||i||p\dot{k}_i||\dot{\omega}||\dot{Y}||\dot{r}||\dot{V} = 0||\bar{M}_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||\bar{r}_j||\bar{V}_j$,

5) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_{k-1}, u_k, \ldots, u_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\check{M}, (i, \check{\omega}, \check{Y}, p\check{k}_i, (\check{V}, \check{z})), p\check{k}_1)$ with $\check{c} = GT[0||p\check{k}_1||i||p\check{k}_i||\check{\omega}||\check{Y}] = u_k$, $\check{r} = RT[p\check{k}_1||i||p\check{k}_i||\check{\omega}||\check{Y}||\check{c}]$, $\check{h} = HT[0||\check{M}||p\check{k}_1||i||p\check{k}_i||\check{\omega}||\check{Y}||\check{r}||\check{V}] = u_j$, $0||p\check{k}_1||i||p\check{k}_i||\check{\omega}||\check{Y} = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||\check{M}||p\check{k}_1||i||p\check{k}_i||\check{\omega}||\check{Y}||\check{r}||\check{V} = 0||\check{M}_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||\check{r}_j||\check{V}_j$, and

6) in the execution of $\boldsymbol{W}((p,q,g,X), s_1, \ldots, s_{k-1}, u_k, \ldots, u_{j-1}, u'_j, \ldots, u'_\gamma; \rho)$, $\boldsymbol{A}$ outputs a valid forgery $(\tilde{M}, (i, \tilde{\omega}, \tilde{Y}, p\tilde{k}_i, (\tilde{V}, \tilde{z})), p\tilde{k}_1)$ with $\tilde{c} = GT[0||p\tilde{k}_1||i||p\tilde{k}_i||\tilde{\omega}||\tilde{Y}] = u_k$, $\tilde{r} = RT[p\tilde{k}_1||i||p\tilde{k}_i||\tilde{\omega}||\tilde{Y}||\tilde{c}]$, $\tilde{h} = HT[0||\tilde{M}||p\tilde{k}_1||i||p\tilde{k}_i||\tilde{\omega}||\tilde{Y}||\tilde{r}||\tilde{V}] = u'_j$, $0||p\tilde{k}_1||i||p\tilde{k}_i||\tilde{\omega}||\tilde{Y} = 0||pk_{1,k}||i||pk_{i,k}||\omega_k||Y_k$, and $0||\tilde{M}||p\tilde{k}_1||i||p\tilde{k}_i||\tilde{\omega}||\tilde{Y}||\tilde{r}||\tilde{V} = 0||\check{M}_j||pk_{1,j}||i||pk_{i,j}||\omega_j||Y_j||\check{r}_j||\check{V}_j$.

From 1) and 2), it follows that $\hat{M} = M$, $p\hat{k}_1 = pk_1$, $p\hat{k}_i = pk_i$, $\hat{\omega} = \omega$, $\hat{Y} = Y$, $\hat{r} = r$, $\hat{V} = V$, $\hat{c} = c$, $g^z \equiv V \cdot ({pk_i}^r \cdot Y \cdot X^c)^h \pmod{p}$, and $g^{\hat{z}} \equiv V \cdot ({pk_i}^r \cdot Y \cdot X^c)^{\hat{h}} \pmod{p}$. Since $\hat{h} \neq h$, $(h - \hat{h})^{-1}$ exists. Thus,

$$g^{(z-\hat{z})(h-\hat{h})^{-1}} \equiv {pk_i}^r \cdot Y \cdot X^c \pmod{p}. \tag{21}$$

From 3) and 4), it follows that $\dot{M} = \bar{M}$, $p\dot{k}_1 = p\bar{k}_1 = pk_1$, $p\dot{k}_i = p\bar{k}_i = pk_i$, $\dot{\omega} = \bar{\omega} = \omega$, $\dot{Y} = \bar{Y} = Y$, $\dot{r} = \bar{r}$, $\dot{V} = \bar{V}$, $\dot{c} = \bar{c}$, $g^{\bar{z}} \equiv \bar{V} \cdot ({pk_i}^{\bar{r}} \cdot Y \cdot X^{\bar{c}})^{\bar{h}} \pmod{p}$, and $g^{\dot{z}} \equiv \bar{V} \cdot ({pk_i}^{\bar{r}} \cdot Y \cdot X^{\bar{c}})^{\dot{h}} \pmod{p}$. Since $\dot{h} \neq \bar{h}$, $(\bar{h} - \dot{h})^{-1}$ exists. Thus,

$$g^{(\bar{z}-\dot{z})(\bar{h}-\dot{h})^{-1}} \equiv {pk_i}^{\bar{r}} \cdot Y \cdot X^{\bar{c}} \pmod{p}. \tag{22}$$

From 5) and 6), it follows that $\tilde{M} = \check{M}$, $p\tilde{k}_1 = p\check{k}_1 = pk_1$, $p\tilde{k}_i = p\check{k}_i = pk_i$, $\tilde{\omega} = \check{\omega} = \omega$, $\tilde{Y} = \check{Y} = Y$, $\tilde{r} = \check{r}$, $\tilde{V} = \check{V}$, $\tilde{c} = \check{c}$, $g^{\check{z}} \equiv \check{V} \cdot ({pk_i}^{\check{r}} \cdot Y \cdot X^{\check{c}})^{\check{h}} \pmod{p}$, and $g^{\tilde{z}} \equiv \check{V} \cdot ({pk_i}^{\check{r}} \cdot Y \cdot X^{\check{c}})^{\tilde{h}} \pmod{p}$. Since $\tilde{h} \neq \check{h}$, $(\check{h} - \tilde{h})^{-1}$ exists. Thus,

$$g^{(\check{z}-\tilde{z})(\check{h}-\tilde{h})^{-1}} \equiv {pk_i}^{\check{r}} \cdot Y \cdot X^{\check{c}} \pmod{p}. \tag{23}$$

Equations (18), (19) and (20) yield the system of equations solved by $\boldsymbol{F}$, where $g^{x_1} = pk_i$, $g^{x_2} = Y$ and $g^{x_3} = X$. If $r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod{q}$, then the system has a unique solution and $\boldsymbol{F}$ returns the discrete logarithm of X.

Let *frk* be defined as in Lemma C.5. Then,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{F}}(k) &\geq \Pr\left[\, b = 1 \wedge r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \not\equiv 0 \pmod{q} \,\right] \\
&\geq \mathit{frk} - \Pr\left[\, r(\check{c} - \bar{c}) - \bar{r}(\check{c} - c) + \check{r}(\bar{c} - c) \equiv 0 \pmod{q} \,\right] \\
&\geq \mathit{frk} - \frac{1}{q} \,.
\end{aligned}$$

The last equation above follows from the fact that values r, $\hat{r}$, $\bar{r}$, c, $\hat{c}$, $\bar{c}$ are independent and uniformly distributed, according to the definitions of $\boldsymbol{W}$ and $\boldsymbol{MF}_{\boldsymbol{W},5}$.

Applying Lemma C.5, we then have

$$\begin{aligned}
&\Pr\left[\, E_5 \,\right] \\
&\leq \; acc + \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{\gamma^{10} \cdot \mathit{frk}} \; + \; \sqrt[6]{\frac{5 \cdot \gamma^{10}}{q}} \; + \\
&\quad\; \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{(q_G + q_H)^{10}\left(\mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{F}}(\kappa) + 1/q\right)} \; + \; \sqrt[6]{\frac{5 \cdot (q_G + q_H)^{10}}{q}} \; + \\
&\quad\; \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \\
&\leq \; \sqrt[6]{(q_G + q_H)^{10} \cdot \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G}_{\mathrm{dl}},\boldsymbol{F}}(\kappa)} \; + \; \sqrt[6]{\frac{(q_G + q_H)^{10}}{q}} \; + \; \sqrt[6]{\frac{5 \cdot (q_G + q_H)^{10}}{q}} \; + \\
&\quad\; \frac{q_d(q_d - 1 + q_G) + q_{sd}(q_{sd} - 1 + q_G) + q_s(q_s - 1 + q_G) + q_p(q_p - 1 + q_H) + 3}{q} \,.
\end{aligned}$$

The last equation follows from the fact that $\sqrt[6]{a + b} \leq \sqrt[6]{a} + \sqrt[6]{b}$ for any real numbers $a, b \geq 0$. This proves Equation (15).

To complete the proof of Theorem 6.2, we observe that the running times of adversaries $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, and $\boldsymbol{F}$ are approximately $2t_{\boldsymbol{A}}$, $4t_{\boldsymbol{A}}$, $6t_{\boldsymbol{A}}$, and $6t_{\boldsymbol{A}}$, respectively.