How to Leak on Key Updates

Allison Lewko *
University of Texas at Austin
alewko@cs.utexas.edu

Mark Lewko
University of Texas at Austin
mlewko@math.utexas.edu

Brent Waters †
University of Texas at Austin
bwaters@cs.utexas.edu

Abstract

In the continual memory leakage model, security against attackers who can repeatedly obtain leakage is achieved by periodically updating the secret key. This is an appealing model which captures a wide class of side-channel attacks, but all previous constructions in this model provide only a very minimal amount of leakage tolerance *during secret key updates*. Since key updates may happen frequently, improving security guarantees against attackers who obtain leakage during these updates is an important problem. In this work, we present the first cryptographic primitives which are secure against a super-logarithmic amount of leakage during secret key updates. We present signature and public key encryption schemes in the standard model which can tolerate a constant fraction of the secret key to be leaked between updates as well as *a constant fraction of the secret key and update randomness* to be leaked during updates. Our signature scheme also allows us to leak a constant fraction of the entire secret state during signing. Before this work, it was unknown how to tolerate super-logarithmic leakage during updates even in the random oracle model. We rely on subgroup decision assumptions in composite order bilinear groups.

1 Introduction

In defining formal notions of security for cryptographic primitives, cryptographers have traditionally modeled attackers as having only black-box access to the primitive. Secret keys and other secret state are assumed to remain completely hidden (except for what can be learned through this black-box access). These security models fail to capture many real scenarios in which an attacker can gain additional information about secret state through side-channels.

A wide variety of side-channel attacks have been developed. Some measure physical features of cryptographic computation like time and power use, etc. (e.g. [5, 6, 7, 8, 26, 34, 35, 42, 45]), while the cold-boot attack [29] demonstrates that an attacker can learn information about the memory of a machine even after the machine is turned off. Since the class of known side-channel attacks is already quite diverse and very likely to grow, it is a risky and unsatisfying approach to depend upon countermeasures against specific side-channel attacks. It is also potentially dangerous to make overly limiting assumptions about what kind of information an attacker obtains through side-channels and what parts of the secret state it depends upon.

*Supported by a National Defense Science and Engineering Graduate Fellowship.

†Supported by NSF CNS-0716199, CNS-0915361, and CNS-0952692, Air Force Office of Scientific Research (AFO SR) under the MURI award for "Collaborative policies and assured information sharing" (Project PRESIDIO), Department of Homeland Security Grant 2006-CS-001-000001-02 (subaward 641), and the Alfred P. Sloan Foundation.

Recently, several security models have been proposed which model side-channel attacks by allowing the attacker to obtain limited amounts of "leakage" through efficiently computable functions applied to the secret state. More precisely, the traditional security game is modified to allow the attacker one or more opportunities to choose an efficiently computable function f with suitably bounded output and learn the value of f applied to the secret key and/or other parts of the secret state. Under these models, there has been much progress in developing cryptographic schemes which are resilient to various kinds and amounts of leakage.

In the bounded leakage model, the attacker may choose an arbitrary efficiently computable leakage function f, but it is assumed that the total leakage occurring throughout the lifetime of the system is bounded. There are many constructions of leakage-resilient primitives in this model, including stream ciphers, symmetric key encryption, signatures, public key encryption, and identity-based encryption [1, 2, 3, 11, 15, 33, 41]. However, assuming that leakage is bounded throughout the lifetime of the system seems unrealistic, since there is no bound a priori on the number of computations that will be performed.

To achieve security in a setting where leakage continually occurs and is not bounded in total, it is clear that one must periodically update the secret key (while keeping the public key fixed). If the secret key remains fixed and an attacker can repeatedly request more and more leakage, the attacker can eventually learn the entire secret key. To achieve security in the presence of continual leakage, many works (e.g. [22, 23, 28, 31, 44]) adopt the assumption introduced by Micali and Reyzin [40] that "only computation leaks information." This assumes that parts of the secret state which are not accessed during a particular computation do not leak during that computation. This assumption may be unrealistic in some scenarios. In particular, it is violated by cold-boot attacks [29]. Also, simply maintaining memory may involve some computation, and memory that is not currently being accessed by a cryptographic computation could still be accessed by the operating system. We would prefer to have a model which is more robust and not threatened by such low-level implementation details.

The Continual Memory Leakage Model Ideally, our goal should be to obtain efficient constructions of cryptographic primitives which are provably secure against the strongest possible class of attackers. To achieve this, we must work with conservative security definitions which do not place unnatural and unnecessary restrictions on attackers that may be violated in practical applications.

The continual memory leakage model, recently introduced by Brakerski et. al. [12] and Dodis et. al. [17], allows continual leakage *without* relying on the Micali-Reyzin assumption. Essentially, this model allows for bounded amounts of leakage to be occurring on all of the secret state all the time. This is a very appealing model that captures the widest class of leakage attacks, but its full generality also makes it challenging to construct schemes which are strongly resilient against *all* of the types of leakage allowed by the model. All of the previous constructions in this model, which include one-way relations, identification schemes, key agreement protocols [17], signatures [10, 12, 17, 39], public key encryption [12], identity based encryption [12, 37], and attribute-based encryption [37], suffer from the same drawback: they can only tolerate an amount of leakage during secret key updates which is logarithmic in terms of the security parameter. Interestingly, this can be viewed as the opposite of the "only computation leaks" assumption, since it is now assumed that (super-logarithmic) leakage occurs only when update computations are *not* taking place.

The Difficulty of Leaking on Updates This very strict limitation on the amount of leakage during updates arises from the following obstacle. Essentially, tolerating a significant amount of leakage during updates requires the simulator to partially "justify" its update to the attacker as being performed honestly according to the update algorithm. At certain stages in the proofs for previous constructions, the simulator is not equipped to do this because it is using part of a challenge to perform the update instead of performing it honestly. Therefore, it does not know a value for the randomness that "explains" its update, and this is an obstruction to satisfying the attacker's leakage request.

As observed by Brakerski et. al. [12] and Waters[1], a logarithmic amount of leakage during updates and the initial key generation can still be obtained by a generic technique of guessing the leakage value and checking the attacker's success rate. The simulator simply tries all potential values for the small number of leaked bits and measures the attacker's success for each one by running the attacker on many random extensions of the game. It then uses the leakage value which appears to maximize the attacker's success rate. This technique cannot extend to more than a logarithmic number of bits, unless one relies on super-polynomial hardness assumptions.

Our Contributions We present a signature scheme and public key encryption scheme in the continual memory leakage model which tolerate a constant fraction of the secret key to be leaked between updates as well as *a constant fraction of the secret key and update randomness* to be leaked during updates. In other words, when the secret key and update randomness together have bit length K, we can allow cK bits of leakage per update where $c > 0$ is a positive constant, independent of the security parameter. Our schemes are proven secure in the standard model, and are the first schemes to achieve more than logarithmic leakage during updates (in the standard *or* random oracle models). We rely only on polynomial hardness assumptions, namely subgroup decision assumptions in composite order bilinear groups. For our signature scheme, updates occur for each signature produced, and there is no additional randomness used for signing. Thus, our scheme achieves strong resilience against memory leakage between updates, leakage on the full state during signing, and leakage on the full state during updates. We do not consider leakage on the initial key generation process, which is only executed once.

Our Techniques In our signature scheme, the secret key consists of group elements in a composite order bilinear group of order N with four distinct prime order subgroups which are orthogonal under the bilinear map. The first subgroup is the only subgroup shared between the secret key and the public parameters, and hence is the only subgroup in which correctness is enforced by the verification algorithm. The second subgroup plays the central role in our proof of leakage resilience, while the third and fourth subgroups provide additional randomization for the secret keys and public key respectively.

The initial secret key will contain group elements which have a structured distribution in the first subgroup (for correctness) and a random distribution in the second and third subgroups. Our update algorithm will choose a random matrix over $\mathbb{Z}_N$ from a certain distribution and apply this matrix in the exponent to "remix" the elements of the secret key. In this way, the new secret key is formed by taking the old secret key elements, raising them to chosen powers, and multiplying together the results to obtain new group elements.

When the updates are chosen from the specified distribution, components from all of the first three subgroups will remain present in the key throughout the lifetime of the system (with all but negligible probability). In this case, it is relatively easy to prove that it is computationally hard for an attacker to produce a forgery that *does not* include components in the second subgroup, since all of the signatures it receives and all of the secret keys its leaks on contain components in this subgroup. It then remains to prove that it is hard for the attacker to produce a forgery that *does* include components in the second subgroup.

[1]This observation is recorded in [17].

If we choose a few of our update matrices from a more restricted distribution, we can cause the components of the secret key in the second subgroup to cancel out at some particular point in the lifetime of the system. By embedding a subgroup decision challenge term in the initial secret key and employing a hybrid argument, we can argue that the exact time this cancelation occurs is computationally hidden. Using our leakage bound, we can also argue that the attacker cannot significantly change its probability of producing a forgery which includes components in the second subgroup.

During some of the hybrid transitions, we will need to change the distribution of a particular update. In these cases, we will need to provide the attacker with leakage from a more restricted update that *looks like* it comes from a properly distributed update. We can accomplish this through Lemma A.1 of [12], which roughly says that "random subspaces are leakage resilient". In our case, this means that when the leakage is not too large, leakage from update matrices chosen under the more restrictive distribution is statistically close to leakage from the proper update distribution.

Our hybrid argument ends with a game where the secret key elements are initialized to have *no components* in the second subgroup. In this setting, it is relatively easy to show that the attacker cannot produce a forgery which has any components in the second subgroup. Since the attacker's probability of producing such a forgery has changed only negligibly through each transition of the hybrid argument, we can conclude that the attacker cannot produce *any* forgeries for our original game with non-negligible probability, and security is proved.

For our PKE scheme, we use essentially the same approach. We start with secret keys and a ciphertext which have no components in the second subgroup and gradually move to a game where all of the secret keys as well as the ciphertext have random components in the second subgroup. Our techniques have some features in common with the dual system encryption methodology introduced by Waters [46], as well as the dual form framework for signatures presented in [27].

Independent Work on "Fully Leakage-Resilient Signatures" In [33], Katz and Vaikuntanathan introduce the terminology "fully leakage-resilient signatures" to refer to signatures in the bounded leakage model which are resilient against leakage which can depend on the *entire* secret state of the signer (i.e. the secret key as well as any randomness used to sign). They provide a one-time signature scheme which is fully leakage-resilient (in the standard model). In this scheme, the signing algorithm is deterministic, so the only secret state of the signer is the secret key itself. They also construct a scheme which is fully leakage-resilient in the random oracle model.

In the continual memory leakage model, [12] and [17] previously constructed signatures which are resilient to continual leakage on the secret key only (in the standard model), as well as signatures resilient to leakage on the secret key and randomness used during signing in the random oracle model. Even in the random oracle model, they allow only logarithmic leakage during updates.

Two concurrent works [10, 39] have presented signature schemes in the standard model that are resilient to leakage on the secret key and the randomness used during signing. The techniques used in these works are quite different from ours. The work of Boyle, Segev, and Wichs introduces the concept of an all-lossy-but-one PKE scheme [10] and combines this with statistical non-interactive witness-indistinguishable arguments. The work of Malkin, Teranishi, Vahlis, and Yung [39] introduces independent pre-image resistant hash functions and also employs Groth-Sahai proofs. The resulting schemes can tolerate leakage up to a $1 - o(1)$ fraction of the secret key length between updates in the continual leakage setting, and do not require updates to be performed for every signature. However, these schemes can still only tolerate a logarithmic number of bits leaked during each update.

While there were prior techniques for achieving resilience against full leakage during signing in the random oracle model, to the best of our knowledge there are no prior techniques for achieving super-logarithmic leakage during updates in the standard or random oracle models. A random oracle does not seem to be helpful in allowing leakage during updates, since updates must preserve some structure and random oracles do not provide this.

1.1 Other Related Work

Exposure-resilient cryptography (e.g. [13, 19, 32]) considered attackers able to learn bounded subsets of the bits representing the secret key, while [30] considered attackers able to learn the values of a subset of wires for a circuit implementing a computation, and [24] considered leakage functions belonging to a low complexity class. The work of [43] constructs pseudorandom generators resistent to certain kinds of naturally occurring leakage.

The bounded leakage model was introduced in [1] and used in many subsequent works (e.g. [2, 3, 11, 15, 33, 41]). Several variations on this model have been considered. For example, the bounded retrieval model was studied in [2, 3, 14, 16, 20, 21]. The work [18] considers the class of leakage functions which are computationally hard to invert.

Several leakage-resilient constructions have been given under the Micali-Reyzin assumption that "only computation leaks information", including stream ciphers [22, 44] and signatures [23]. One can view the work of [22] as updating a seed, but the techniques employed here are tied to the Micali-Reyzin assumption. More generally, the works [28, 31] provide a method for making any cryptographic algorithm secure against continual leakage - relying on the Micali-Reyzin assumption as well as a simple, completely non-leaking hardware device.

A few recent works [15, 37] have employed the dual system encryption methodology introduced by Waters [46] in the leakage setting. Dual system encryption was designed as a tool for proving adaptive security for advanced functionalities (e.g. IBE, HIBE, ABE [36, 38, 46]), but it extends quite naturally to provide leakage resilience as well, as shown in [37]. However, this work does not provide resilience against super-logarithmic leakage during updates in the continual leakage setting.

1.2 Organization

In Section 2, we give the necessary background, our formal security definitions, and our complexity assumptions. In Section 3, we present our signature scheme. In Section 4, we present our PKE scheme. In Section 5, we prove security for our signature scheme. In Section 6, we discuss the leakage parameters we obtain, extensions of our techniques, and remaining open problems. In Appendix D, we prove security for our PKE scheme.

2 Background

2.1 Signature Schemes

A signature scheme typically consists of three algorithms: KeyGen, Sign, and Verify. In the continual leakage setting, we require an additional algorithm Update which updates the secret key. Note that the verification key remains unchanged.

KeyGen$(\lambda) \rightarrow \mathrm{VK}, \mathrm{SK}_0$ The key generation algorithm takes in the security parameter, λ, and outputs a secret key SK_0 and a public verification key VK.

Sign$(m, \mathrm{SK}_i) \rightarrow \sigma$ The signing algorithm[2] takes in a message m and a secret key SK_i, and outputs a signature σ.

Verify$(\mathrm{VK}, \sigma, m) \rightarrow \{True, False\}$ The verification algorithm takes in the verification key VK, a signature σ, and a message m. It outputs either "True" or "False".

Update$(\mathrm{SK}_{i-1}) \rightarrow \mathrm{SK}_i$ The update algorithm takes in a secret key SK_{i-1} and produces a new secret key SK_i for the *same* verification key.

Correctness The signature scheme satisfies correctness if $\mathrm{Verify}(\mathrm{VK}, \sigma, m)$ outputs "True" whenever $\mathrm{VK}, \mathrm{SK}_0$ is produced by KeyGen, and σ is produced by $\mathrm{Sign}(m, \mathrm{SK}_i)$ for some SK_i obtained by calls to Update, starting with SK_0. (If the verification algorithm is randomized, we may relax this requirement to hold with all but negligible probability.)

2.2 Security Definition for Signatures

We define leakage-resilient security for signatures in terms of the following game between a challenger and an attacker (this extends the usual notion of existential unforgeability to our leakage setting). The game is parameterized by two values: the security parameter λ, and the parameter ℓ which controls the amount of leakage allowed. For the sake of simplicity, we assume that the signing algorithm calls the update algorithm on each invocation. We note that [10, 12, 17, 39] give a more general definition for signatures resilient to continual leakage which does not assume that key updates occur with each signature and allows different leakage parameters for during and between updates. Since updates in our scheme do occur with each signature, we find it more convenient to work with the simplified definition given below.

Setup Phase The game begins with a setup phase. The challenger calls $\mathrm{KeyGen}(\lambda)$ to create the signing key, SK_0, and the verification key, VK. It gives VK to the attacker. *No leakage is allowed in this phase.*

Query Phase The attacker specifies a message, m_1, which it gives to the challenger, along with an efficiently computable leakage function f_1, whose output is at most ℓ bits. The challenger chooses some randomness X_1, updates the secret key (changing it from SK_0 to SK_1), and then signs the message. (The randomness X_1 denotes all randomness used for the update and the signing process.) It gives the attacker the resulting signature, along with $f_1(X_1, \mathrm{SK}_0)$. The attacker then repeats this a polynomial number of times, each time supplying a message m_i and an efficiently computable leakage function f_i whose output is at most ℓ bits[3]. Each time the challenger chooses randomness X_i, updates the secret key from SK_{i-1} to SK_i, and gives the attacker a signature on m_i as well as $f_i(X_i, \mathrm{SK}_{i-1})$.

Forgery Phase The attacker gives the challenger a message, m^*, and a signature σ^* such that m^* has not been previously queried. The attacker wins the game if (m^*, σ^*) passes the verification algorithm using VK.

Definition 1. *We say a signature scheme is ℓ-leakage resilient against continual leakage on memory and computation if any probabilistic polynomial time attacker only has a negligible probability (negligible in λ) of winning the above game.*

[2]In our security definition, we will assume that each invocation of the Sign algorithm calls the Update algorithm.

[3]We assume the output length of each f_i is independent of the input value.

2.3 Public Key Encryption

A Public Key Encryption (PKE) scheme typically consists of three algorithms: KeyGen, Encrypt, and Decrypt. In the continual leakage setting, we require an additional algorithm Update which updates the secret key. Note that the public key remains unchanged.

KeyGen$(\lambda) \to \mathrm{PK}, \mathrm{SK}_0$ The key generation algorithm takes in the security parameter λ and outputs a public key PK and a secret key SK_0.

Encrypt$(M, \mathrm{PK}) \to \mathrm{CT}$ The encryption algorithm takes in a message M and a public key PK. It outputs a ciphertext CT.

Decrypt$(\mathrm{CT}, \mathrm{SK}_i) \to M$ The decryption algorithm takes in a ciphertext CT and a secret key SK_i. It outputs a message M.

Update$(\mathrm{SK}_{i-1}) \to \mathrm{SK}_i$ The update algorithm takes in a secret key SK_{i-1} and produces a new secret key SK_i for the *same* public key.

Correctness The PKE scheme satisfies correctness if $\mathrm{Decrypt}(\mathrm{CT}, \mathrm{SK}_i) = M$ with all but negligible probability whenever $\mathrm{PK}, \mathrm{SK}_0$ is produced by KeyGen, SK_i is obtained by calls to Update on previously obtained secret keys (starting with SK_0), and CT is produced by $\mathrm{Encrypt}(M, \mathrm{PK})$.

2.4 Security Definition for PKE

We define leakage-resilient security for PKE schemes in terms of the following game between a challenger and an attacker (this extends the usual notion of semantic security to our leakage setting). We let λ denote the security parameter, and the parameter ℓ controls the amount of leakage allowed.

Setup Phase The game begins with a setup phase. The challenger calls $\mathrm{KeyGen}(\lambda)$ to create the initial secret key SK_0 and public key PK. It gives PK to the attacker. *No leakage is allowed in this phase.*

Query Phase The attacker specifies an efficiently computable leakage function f_1, whose output is at most ℓ bits. The challenger chooses some randomness X_1, updates the secret key (changing it from SK_0 to SK_1), and then gives the attacker $f_1(X_1, \mathrm{SK}_0)$. The attacker then repeats this a polynomial number of times, each time supplying an efficiently computable leakage function f_i whose output is at most ℓ bits[4]. Each time, the challenger chooses randomness X_i, updates the secret key from SK_{i-1} to SK_i, and gives the attacker $f_i(X_i, \mathrm{SK}_{i-1})$.

Challenge Phase The attacker chooses two messages M_0, M_1 which it gives to the challenger. The challenger chooses a random bit $b \in \{0, 1\}$, encrypts M_b, and gives the resulting ciphertext to the attacker. The attacker then outputs a guess b' for b. The attacker wins the game if $b = b'$. We define the advantage of the attacker in this game as $\left|\frac{1}{2} - Pr[b = b']\right|$.

Definition 2. *We say a Public Key Encryption scheme is ℓ-leakage resilient against continual leakage on memory and computation if any probabilistic polynomial time attacker only has a negligible advantage (negligible in λ) in the above game.*

[4]We again assume the output length of each f_i is independent of the input value.

2.5 Composite Order Bilinear Groups

Our schemes will be constructed in composite order bilinear groups, which were first introduced in [9]. We let $\mathcal{G}$ denote a group generator, i.e. an algorithm which takes a security parameter λ as input and outputs a description of a bilinear group G. For our purposes, we define $\mathcal{G}$'s output as (N, G, G_T, e), where $N = p_1p_2p_3p_4$ is a product of four distinct primes, G and G_T are cyclic groups of order N, and $e : G^2 \to G_T$ is a map such that:

1. (Bilinear) $\forall g, h \in G,\ a, b \in \mathbb{Z}_N,\ e(g^a, h^b) = e(g, h)^{ab}$
2. (Non-degenerate) $\exists g \in G$ such that $e(g, g)$ has order N in G_T.

The group operations in G and G_T and the map e are computable in polynomial time with respect to λ, and the group descriptions of G and G_T include a generator of each group. We let G_{p_1}, G_{p_2}, G_{p_3}, and G_{p_4} denote the subgroups of order p_1, p_2, p_3, and p_4 in G respectively. We note that these subgroups are "orthogonal" to each other under the bilinear map e: i.e. if $h_i \in G_{p_i}$ and $h_j \in G_{p_j}$ for $i \neq j$, then $e(h_i, h_j)$ is the identity element in G_T. If g_1 generates G_{p_1}, g_2 generates G_{p_2}, g_3 generates G_{p_3}, and g_4 generates G_{p_4}, then every element h of G can be expressed as $g_1^w g_2^x g_3^y g_4^z$ for some values $w, x, y, z \in \mathbb{Z}_N$. We will refer to g_1^w as the "G_{p_1} part of h", for example.

2.6 Our Complexity Assumptions

Our complexity assumptions are all instances of the Generalized Subgroup Decision Assumption described in [4]. In a bilinear group of order $N = p_1p_2 \dots p_n$, there is a subgroup of order $\prod_{i \in S} p_i$ for each subset $S \subseteq \{1, \dots, n\}$. We let S_0, S_1 denote two such subsets/subgroups. The Generalized Subgroup Decision Assumption says that it is hard to distinguish a random element from the subgroup S_0 from a random element of S_1 when one is only given random elements from subgroups Z_i which satisfy either $S_0 \cap Z_i = \emptyset = S_1 \cap Z_i$ or $S_0 \cap Z_i \neq \emptyset \neq S_1 \cap Z_i$ (when viewed as subsets of $\{1, \dots, n\}$). We note that these conditions prevent an attacker from distinguishing elements of S_0 from elements of S_1 by pairing with the Z_i elements using the bilinear map. These assumptions hold in the generic group model for composite order bilinear groups.

In the formal descriptions of our assumptions below, we let $G_{p_1p_2}$ (e.g.) denote the subgroup of order p_1p_2 in G. We use the notation $X \xleftarrow{R} Z$ to express that X is chosen uniformly randomly from the finite set Z.

Assumption 1 Given a group generator $\mathcal{G}$, we define the following distribution:

$$\mathbb{G} = (N = p_1p_2p_3p_4, G, G_T, e) \xleftarrow{R} \mathcal{G},$$

$$X_1, Y_1 \xleftarrow{R} G_{p_1},\ Y_2, Z_2 \xleftarrow{R} G_{p_2},\ g_3, Y_3, Z_3 \xleftarrow{R} G_{p_3}, g_4, X_4 \xleftarrow{R} G_{p_4}$$

$$D = (\mathbb{G}, g_3, g_4, X_1X_4, Y_1Y_2Y_3, Z_2Z_3),$$

$$T_1 \xleftarrow{R} G_{p_2p_4},\ T_2 \xleftarrow{R} G_{p_1p_2p_4}.$$

We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 1 to be:

$$Adv1_{\mathcal{G},\mathcal{A}}(\lambda) := \big|Pr[\mathcal{A}(D, T_1) = 1] - Pr[\mathcal{A}(D, T_2) = 1]\big|.$$

We say that $\mathcal{G}$ satisfies Assumption 1 if $Adv1_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any PPT algorithm $\mathcal{A}$.

Assumption 2 Given a group generator $\mathcal{G}$, we define the following distribution:

$$\mathbb{G} = (N = p_1p_2p_3p_4, G, G_T, e) \xleftarrow{R} \mathcal{G},$$

$$g \xleftarrow{R} G_{p_1},\ g_3 \xleftarrow{R} G_{p_3},\ g_4 \xleftarrow{R} G_{p_4}$$

$$D = (\mathbb{G}, g, g_3, g_4),$$

$$T_1 \xleftarrow{R} G_{p_1},\ T_2 \xleftarrow{R} G_{p_1p_2}.$$

We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 2 to be:

$$Adv2_{\mathcal{G},\mathcal{A}}(\lambda) := \big|Pr[\mathcal{A}(D,T_1) = 1] - Pr[\mathcal{A}(D,T_2) = 1]\big|.$$

We say that $\mathcal{G}$ satisfies Assumption 2 if $Adv2_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any PPT algorithm $\mathcal{A}$.

Assumption 3 Given a group generator $\mathcal{G}$, we define the following distribution:

$$\mathbb{G} = (N = p_1p_2p_3p_4, G, G_T, e) \xleftarrow{R} \mathcal{G},$$

$$g, X_1 \xleftarrow{R} G_{p_1},\ X_2, Y_2 \xleftarrow{R} G_{p_2},\ g_3, Y_3 \xleftarrow{R} G_{p_3},\ g_4 \xleftarrow{R} G_{p_4}$$

$$D = (\mathbb{G}, g, g_3, g_4, X_1X_2, Y_2Y_3),$$

$$T_1 \xleftarrow{R} G_{p_1p_3},\ T_2 \xleftarrow{R} G_{p_1p_2p_3}.$$

We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 3 to be:

$$Adv3_{\mathcal{G},\mathcal{A}}(\lambda) := \big|Pr[\mathcal{A}(D,T_1) = 1] - Pr[\mathcal{A}(D,T_2) = 1]\big|.$$

We say that $\mathcal{G}$ satisfies Assumption 3 if $Adv3_{\mathcal{G},\mathcal{A}}(\lambda)$ is a negligible function of λ for any PPT algorithm $\mathcal{A}$.

2.7 Notation

In our constructions, we will often represent tuples of group elements as a base element raised to a vector. For $\vec{r} \in \mathbb{Z}_N^n$, we use the notation $g^{\vec{r}}$, for example, to denote the n-tuple of group elements

$$g^{\vec{r}} := (g^{r_1}, \ldots, g^{r_n}).$$

We use $g^{\vec{r}} g_2^{\vec{c}}$ (e.g.) to denote the n-tuple of group elements formed by componentwise multiplication:

$$g^{\vec{r}} g_2^{\vec{c}} := (g^{r_1} g_2^{c_1}, \ldots, g^{r_n} g_2^{c_n}).$$

2.8 Min-Entropy, Statistical Distance, and Mod N Arithmetic

We let X denote a random variable which takes values in a finite set. We define the min-entropy of X, denoted $H_\infty(X)$, as follows:

$$H_\infty(X) := -\log(\max_x Pr[X = x]).$$

We similarly define the min-entropy of X conditioned on an event E as follows:

$$H_\infty(X|E) := -\log(\max_x Pr[X = x|E]).$$

We will require the following standard lemma about min-entropy (e.g. [33]):

Lemma 3. *Let X be a random variable with min-entropy h and let f be an arbitrary function with range $\{0,1\}^\ell$. For any $\tau \in [0, h-\ell]$, we define the set*

$$V_\tau = \{v \in \{0,1\}^\ell | H_\infty(X|v = f(X)) \leq h - \ell - \tau\}.$$

Then:

$$Pr[f(X) \in V_\tau] \leq 2^{-\tau}.$$

For two random variables X_1, X_2 taking values in the same finite set, we define the statistical distance between them to be:

$$dist(X_1, X_2) := \frac{1}{2} \sum_x |Pr[X_1 = x] - Pr[X_2 = x]|.$$

Throughout our security proofs, we will be implicitly using the Chinese Remainder Theorem. In particular, this theorem implies that choosing a random value modulo N is equivalent to choosing random values modulo p_1, p_2, p_3 and p_4 independently. This means that if we have a generator of a subgroup of G and we know N, we can sample a new, uniformly random element of the subgroup by raising our generator to a random exponent modulo N. In working over $\mathbb{Z}_N$, we will routinely ignore the negligible probability that we ever encounter a nonzero, non-invertible element of $\mathbb{Z}_N$ while doing computations on randomly chosen elements. We will also ignore the negligible probability that $\leq d$ vectors over $\mathbb{Z}_N$ chosen uniformly at random from a space of dimension d are linearly dependent.

3 Our Signature Scheme

We now present our leakage-resilient signature scheme. The message space for our scheme is $\mathbb{Z}_N$. Our secret key will consist of several elements in a composite order bilinear group of order N, and we will update it by "mixing" these elements in a structured way. In particular, new elements will be obtained by raising the current elements to exponents specified by a matrix and multiplying the results. This matrix will be chosen freshly at random from a certain distribution for each update. To maintain correctness, we must preserve the relevant relationships between secret key element exponents in the G_{p_1} subgroup. This is the only subgroup shared between the secret key and the verification key: the G_{p_2} and G_{p_3} subgroups provide randomization for the secret key, while the G_{p_4} subgroup is used to blind the G_{p_1} elements of the verification key. Correctness in the G_{p_1} subgroup is maintained by applying the *same* update matrix independently to several "columns" of elements, which preserves the required relationships across columns. One can alternatively view the secret key as a single (larger) vector of group elements where correctness with the fixed verification key is maintained as long as the exponent vector belongs to a fixed subspace over $\mathbb{Z}_{p_1}$. Our update matrices will be applied in a way that ensures each new secret key will have an exponent vector also in this subspace.

3.1 Construction

Our signature scheme consists of four algorithms, KeyGen, Update, Sign, and Verify. Note that Sign calls Update on each invocation. The parameter n represents a natural number that is ≥ 9.

KeyGen(λ) $\rightarrow \mathrm{SK}_0, \mathrm{VK}$ The key generation algorithm chooses an appropriate composite order bilinear group G, whose order $N = p_1 p_2 p_3 p_4$ is a product of four distinct primes. It chooses g, u, h randomly from G_{p_1} and R, R', R'', R''' randomly from G_{p_4}. It sets:

$$\mathrm{VK} = \{N, G, R, gR',\ uR'',\ hR'''\}.$$

It also chooses g_2 randomly from G_{p_2}, g_3 randomly from G_{p_3}, and random vectors $\vec{r} = (r_1, \ldots, r_n)$, $\vec{c} = (c_1, \ldots, c_n)$, $\vec{d} = (d_1, \ldots, d_n)$, $\vec{f} = (f_1, \ldots, f_n)$, $\vec{x} = (x_1, \ldots, x_n)$, $\vec{y} = (y_1, \ldots, y_n)$, $\vec{z} = (z_1, \ldots, z_n) \in \mathbb{Z}_N^n$. We recall that the notation $g^{\vec{r}}$ denotes the n-tuple of group elements $(g^{r_1}, \ldots, g^{r_n})$ and $g^{\vec{r}} g_2^{\vec{c}}$ denotes the n-tuple of group elements formed by componentwise multiplication: $(g^{r_1} g_2^{c_1}, \ldots, g^{r_n} g_2^{c_n})$. We let $\vec{S}_0 = (S_{1,0}, \ldots, S_{n,0})$, $\vec{U}_0 = (U_{1,0}, \ldots, U_{n,0})$, and $\vec{H}_0 = (H_{1,0}, \ldots, H_{n,0})$ be n-tuples of group elements defined as follows:

$$\vec{S}_0 := g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}},\ \vec{U}_0 := u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}},\ \vec{H}_0 := h^{\vec{r}} g_2^{\vec{f}} g_3^{\vec{z}}.$$

The secret key is $\mathrm{SK}_0 := \{\vec{S}_0, \vec{U}_0, \vec{H}_0\}$ (this contains $3n$ group elements).

Update(SK_{i-1}) $\rightarrow \mathrm{SK}_i$ The secret key update algorithm picks two random vectors $\vec{a} = (a_1, \ldots, a_{n-1})$ and $\vec{b} = (b_1, \ldots, b_{n-1})$ from $\mathbb{Z}_N^{n-1}$ and computes the new secret key $\mathrm{SK}_i = \{\vec{S}_i, \vec{U}_i, \vec{H}_i\}$ from the old secret key as follows:

$$S_{1,i} := S_{1,i-1} \cdot S_{n,i-1}^{b_1},\ U_{1,i} := U_{1,i-1} \cdot U_{n,i-1}^{b_1},\ H_{1,i} := H_{1,i-1} \cdot H_{n,i-1}^{b_1},$$

$$S_{2,i} := S_{2,i-1} \cdot S_{n,i-1}^{b_2},\ U_{2,i} := U_{2,i-1} \cdot U_{n,i-1}^{b_2},\ H_{2,i} := H_{2,i-2} \cdot H_{n,i-1}^{b_2},$$

$$\vdots$$

$$S_{n-1,i} := S_{n-1,i-1} \cdot S_{n,i-1}^{b_{n-1}},\ U_{n-1,i-1} \cdot U_{n,i-1}^{b_{n-1}},\ H_{n-1,i} := H_{n-1,i-1} \cdot H_{n,i-1}^{b_{n-1}},$$

$$S_{n,i} := S_{1,i-1}^{a_1} \cdot S_{2,i-1}^{a_2} \cdots S_{n-1,i-1}^{a_{n-1}} \cdot S_{n,i-1}^{\vec{a}\cdot\vec{b}},$$

$$U_{n,i} := U_{1,i-1}^{a_1} \cdot U_{2,i-1}^{a_2} \cdots U_{n-1,i-1}^{a_{n-1}} \cdot U_{n,i-1}^{\vec{a}\cdot\vec{b}},$$

$$H_{n,i} := H_{1,i-1}^{a_1} \cdot H_{2,i-1}^{a_2} \cdots H_{n-1,i-1}^{a_{n-1}} \cdot H_{n,i-1}^{\vec{a}\cdot\vec{b}}.$$

This should be thought of as multiplying on the left by the following matrix in the exponent (separately for $\vec{S}, \vec{U}, \vec{H}$):

$$A = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & b_1 \\ 0 & 1 & 0 & \ldots & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & b_{n-1} \\ a_1 & a_2 & a_3 & \cdots & a_{n-1} & \vec{a}\cdot\vec{b} \end{pmatrix}$$

In other words, when we begin with $\vec{S}_0 := g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}}$, $\vec{U}_0 := u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}}$, and $\vec{H}_0 = h^{\vec{r}} g_2^{\vec{f}} g_3^{\vec{z}}$, then after applying a single update with matrix A, we will have

$$\vec{S}_1 = g^{A\vec{r}} g_2^{A\vec{c}} g_3^{A\vec{x}},$$

$$\vec{U}_1 = u^{A\vec{r}} g_2^{A\vec{d}} g_3^{A\vec{y}},$$

$$\vec{H}_1 = h^{A\vec{r}} g_2^{A\vec{f}} g_3^{A\vec{z}}.$$

We note that A is an $n \times n$ matrix of rank $n-1$ (since the last row is a linear combination of the previous rows). With all but negligible probability, a product of many such matrices chosen independently will also be rank $n-1$ (so we are not losing rank with the application of each new update, we will remain at rank $n-1$)[5]. We also note that after applying each update, the secret key remains the same size.

In our proofs, we will be choosing many update matrices which we will denote by $A_1, A_2, \dots$ (where A_i is the matrix used for the i^{th} update). We will let $\vec{a}_i = (a_1^i, \dots, a_{n-1}^i)$ denote the first $n-1$ entries of the last row of A_i and $\vec{b}_i = (b_1^i, \dots, b_{n-1}^i)$ denote the first $n-1$ entries of the last column of A_i.

Sign$(m, \mathrm{SK}_{i-1}) \rightarrow \sigma$ The signing algorithm first calls Update(SK_{i-1}) to obtain SK_i. It produces the signature σ as:

$$\sigma := (\sigma_1,\ \sigma_2) = (U_{1,i}^m H_{1,i},\ S_{1,i}).$$

We note that the only randomness used here during signing is the random choice of the update matrix.

Verify$(\mathrm{VK}, \sigma, m) \rightarrow \{True,\ False\}$ The verification algorithm checks that

$$e(\sigma_1, gR') = e(\sigma_2, (uR'')^m (hR''')) \neq 1,$$

and that

$$e(\sigma_1, R) = e(\sigma_2, R) = 1.$$

If both checks pass, the algorithm outputs "True". Otherwise, it outputs "False." (This second check ensures that there are no elements of G_{p_4} appearing in σ_1, σ_2.)

Correctness To verify correctness for our scheme, we note that the update algorithm preserves the relevant relationships between the G_{p_1} parts of the secret key. This means that for any secret key SK_i obtained by applying the update algorithm an arbitrary number of times, the G_{p_1} parts of $\vec{S}_i$, $\vec{U}_i$, and $\vec{H}_i$ will remain of the form $g^{\vec{r'}}, u^{\vec{r'}}, h^{\vec{r'}}$ for some $\vec{r'} \in \mathbb{Z}_N^n$. Thus, if (σ_1, σ_2) is a signature produced from Sign(m, SK_i), we will have:

$$\sigma_1 = (u^m h)^{r'} g_2^{s_2} g_3^{s_3},\ \ \sigma_2 = g^{r'} g_2^{t_2} g_3^{t_3}$$

for some values $r', s_2, s_3, t_2, t_3 \in \mathbb{Z}_N$. Then:

$$e(\sigma_1, gR') = e(u^m h, g)^{r'} = e(\sigma_2, (uR'')^m (hR''')),$$

and both of σ_1, σ_2 are orthogonal to G_{p_4} under the bilinear map e, so this signature verifies correctly.

3.2 Security

In Section 5, we prove the following security theorem for our signature scheme:

Theorem 4. *Under Assumptions 1, 2, and 3, when ℓ is at most the minimum of $\frac{1}{3}(\log(p_2) - 2\delta)$ and $(n-8)\log(p_j) - 2\delta$ for all primes p_j dividing N (where δ is set so that $2^{-\delta}$ is negligible), our signature scheme is ℓ-leakage resilient against continual leakage on memory and computation, as defined by Definition 1.*

[5]See Corollary 14 for a proof of this.

4 Our PKE Scheme

We now present our leakage-resilient public key encryption scheme. As in our signature scheme, the secret key will consist of group elements in a composite order group. We will again update the secret key by applying a matrix, chosen from the same distribution as before. Our message space will now be $\{0, 1\}$ (i.e. we will encrypt one bit at a time). We will maintain correctness with the public key by applying the same matrix to each of three columns of group elements, which preserves the ratio of exponents across columns.

4.1 Construction

Our PKE scheme consists of four algorithms, KeyGen, Update, Encrypt, and Decrypt. The parameter n represents a natural number that is ≥ 9.

KeyGen(λ) $\rightarrow \mathrm{PK}, \mathrm{SK}_0$ The key generation algorithm chooses an appropriate composite order bilinear group G, whose order $N = p_1p_2p_3p_4$ is a product of four distinct primes. It then chooses a random element $g \in G_{p_1}$, random exponents $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{Z}_N$, a random element $g_3 \in G_{p_3}$, and random elements $R, R', R'', R''' \in G_{p_4}$. It sets the public key as:

$$\mathrm{PK} := \{N, G, R, g^{\alpha_1}R', g^{\alpha_2}R'', g^{\alpha_3}R'''\}.$$

It then chooses random vectors $\vec{r}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$, as well as a random vector $\vec{\eta} = (\eta_1, \eta_2, \eta_3) \in \mathbb{Z}_N^3$ subject to the constraint that $\vec{\eta} \cdot \vec{\alpha} = 0$ (where $\vec{\alpha} = (\alpha_1, \alpha_2, \alpha_3)$). It forms $\vec{S}_0 = (S_{1,0}, \dots, S_{n,0})$ as $g^{\eta_1 \vec{r}} g_3^{\vec{x}}$, forms $\vec{U}_0 = (U_{1,0}, \dots, U_{n,0})$ as $g^{\eta_2 \vec{r}} g_3^{\vec{y}}$, and forms $\vec{H}_0 = (H_{1,0}, \dots, H_{n,0})$ as $g^{\eta_3 \vec{r}} g_3^{\vec{z}}$. It sets the initial secret key as:

$$\mathrm{SK}_0 := \{\vec{S}_0,\ \vec{U}_0,\ \vec{H}_0\}.$$

We note that the secret key contains $3n$ group elements.

Update$(\mathrm{SK}_{i-1}) \rightarrow \mathrm{SK}_i$ The secret key update algorithm is the same as the update algorithm for our signature scheme. It picks two random vectors $\vec{a} = (a_1, \dots, a_{n-1})$ and $\vec{b} = (b_1, \dots, b_{n-1})$ from $\mathbb{Z}_N^{n-1}$ and computes the new secret key $\mathrm{SK}_i = \{\vec{S}_i, \vec{U}_i, \vec{H}_i\}$ from the old secret key as follows:

$$\begin{gathered}
S_{1,i} := S_{1,i-1} \cdot S_{n,i-1}^{b_1},\ U_{1,i} := U_{1,i-1} \cdot U_{n,i-1}^{b_1}, \\
S_{2,i} := S_{2,i-1} \cdot S_{n,i-1}^{b_2},\ U_{2,i} := U_{2,i-1} \cdot U_{n,i-1}^{b_2}, \\
\vdots \\
S_{n-1,i} := S_{n-1,i-1} \cdot S_{n,i-1}^{b_{n-1}},\ U_{n-1,i-1} \cdot U_{n,i-1}^{b_{n-1}}, \\
S_{n,i} := S_{1,i-1}^{a_1} \cdot S_{2,i-1}^{a_2} \cdots S_{n-1,i-1}^{a_{n-1}} \cdot S_{n,i-1}^{\vec{a} \cdot \vec{b}}, \\
U_{n,i} := U_{1,i-1}^{a_1} \cdot U_{2,i-1}^{a_2} \cdots U_{n-1,i-1}^{a_{n-1}} \cdot U_{n,i-1}^{\vec{a} \cdot \vec{b}},
\end{gathered}$$

This should be thought of as multiplying on the left by the following matrix in the exponent:

$$A = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & b_1 \\ 0 & 1 & 0 & \dots & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & b_{n-1} \\ a_1 & a_2 & a_3 & \cdots & a_{n-1} & \vec{a} \cdot \vec{b} \end{pmatrix}$$

We note that this is an $n \times n$ matrix of rank $n - 1$ (since the last row is a linear combination of the previous rows).

Encrypt$(M, \mathrm{PK}) \rightarrow \mathrm{CT}$ The encryption algorithm takes in a message M which is a single bit (i.e. $M \in \{0,1\}$). The algorithm chooses three random exponents $s, t, v \in \mathbb{Z}_N$. When $M = 0$, it sets $C_1 := (g^{\alpha_1} R')^s$, $C_2 := (g^{\alpha_2} R'')^s R^t$, $C_3 := (g^{\alpha_3} R''')^s R^v$. When $M = 1$, it sets $C_1 := (g^{\alpha_1} R')^s$, $C_2 := (g^{\alpha_1} R')^t$, $C_3 := (g^{\alpha_1} R')^v$. The ciphertext is $\mathrm{CT} := \{C_1, C_2, C_3\}$.

When $M = 0$, the components of the three ciphertext elements in G_{p_1} will be $(g^{\alpha_1 s}, g^{\alpha_2 s}, g^{\alpha_3 s})$, and the components in G_{p_4} will be uniformly random. When $M = 1$, the components in G_{p_1} and G_{p_4} will both be uniformly random.

Decrypt$(\mathrm{CT}, \mathrm{SK}_i)$ The decryption algorithm checks if

$$e(C_1, S_{1,i}) e(C_2, U_{1,i}) e(C_3, H_{1,i}) = 1,$$

where 1 denotes the identity element in G_T. If this equality holds, it outputs $M = 0$. If it does not hold, it outputs $M = 1$.

Correctness If SK_i is obtained from SK_0 by applying our update algorithm an arbitrary number of times, the G_{p_1} parts of $\vec{S}_i$, $\vec{U}_i$, and $\vec{H}_i$ will be of the form $g^{\eta_1 \vec{r'}}, g^{\eta_2 \vec{r'}}, g^{\eta_3 \vec{r'}}$ for some vector $\vec{r'} \in \mathbb{Z}_N$ (this form is preserved by the update algorithm). Hence, the G_{p_1} parts of $S_{1,i}$, $U_{1,i}$, and $H_{1,i}$ are equal to $g^{\eta_1 r'}$, $g^{\eta_2 r'}$, and $g^{\eta_3 r'}$ for some $r' \in \mathbb{Z}_N$. If $C_1 := (gR')^s$, $C_2 := (g^{\alpha} R'')^s R^t$, $C_3 := (g^{\beta} R''')^s R^v$ is an encryption of 0 formed by calling Encrypt(0, PK), then

$$e(C_1, S_{1,i}) e(C_2, U_{1,i}) e(C_3, H_{1,i}) = e(g,g)^{sr'\alpha_1\eta_1} e(g,g)^{sr'\alpha_2\eta_2} e(g,g)^{sr'\alpha_3\eta_3} = 1,$$

since $\vec{\alpha} \cdot \vec{\eta} = 0$. In this case, the decryption algorithm will correctly output $M = 0$.

If $C_1 := (g^{\alpha_1} R')^s$, $C_2 := (g^{\alpha_1} R')^t$, $C_3 := (g^{\alpha_1} R')^v$ is an encryption of 1 formed by calling Encrypt(1, PK), then

$$e(C_1, S_{1,i}) e(C_2, U_{1,i}) e(C_3, H_{1,i}) = e(g,g)^{sr'\alpha_1\eta_1} e(g,g)^{tr'\alpha_1\eta_2} e(g,g)^{vr'\alpha_1\eta_3}.$$

With all but negligible probability over the choice of $r', t, s, v, \vec{\alpha}, \vec{\eta}$, we will have

$$r'\alpha_1(s\eta_1 + t\eta_2 + v\eta_3) \neq 0$$

modulo p_1, and the decryption algorithm will correctly output $M = 1$. We do incur a negligible correctness error when $r'\alpha_1(s\eta_1 + t\eta_2 + v\eta_3)$ happens to equal 0 modulo p_1.

4.2 Security

In Appendix D, we prove the following security theorem for our PKE scheme:

Theorem 5. *Under Assumptions 1, 2, and 3, when ℓ is at most the minimum of $\frac{1}{3}(\log(p_2) - 2\delta)$ and $(n-8)\log(p_j) - 2\delta$ for all primes p_j dividing N (where δ is set so that $2^{-\delta}$ is negligible), our PKE scheme is ℓ-leakage resilient against continual leakage on memory and computation, as defined by Definition 2.*

5 Security Proof for Our Signature Scheme

We now prove security for our signature scheme. In the real security game, all of the secret keys and signatures will have components in G_{p_2}. We show that in this setting, an attacker has only a negligible chance of producing a forgery that *does not have* any G_{p_2} parts. The main idea of our security proof is to use a hybrid argument to gradually move to a game where *none* of the secret keys and signatures have any components in G_{p_2}. Essentially, we will employ update matrices which cancel out the G_{p_2} terms in the secret key at progressively earlier stages in the game.

The core technique of our proof is to embed a challenge term T from a subgroup decision problem into the initial secret key. T will be a group element which definitely has G_{p_1} and G_{p_3} parts, and it is the simulator's task to decide if it also has a G_{p_2} part or not. The simulator will choose update matrices which first cancel out the terms in the secret key which definitely have G_{p_2} components, and then will cancel out the instances of T. If this cancelation of T happens during the $i+1$ update, we can use this to move from a game where the G_{p_2} components of the secret key are canceled out at step $i+1$ to a game where they are canceled out at step i.

There are two important subtleties in executing this approach: first, the distribution of the G_{p_2} parts of the initial secret key will *depend upon* the nature of the challenge term T, and second, update matrices capable of canceling terms in the keys must be chosen from a more restricted distribution. We address these subtleties by expanding our sequence of games to include these changes in the initial key distribution and in the timing of the canceling updates in the game definitions.

As we move from game to game, we must argue that the attacker's chance of producing a forgery that does have G_{p_2} parts changes only negligibly: in some cases, we will show this by relying on a computational assumption and a min-entropy argument. In other cases, we will show that the two games are (information-theoretically) indistinguishable in the attacker's view because of the bound on the leakage parameter ℓ. For these parts of the proof, we will rely on a useful lemma from [12], which roughly says that random subspaces are leakage resilient. This lemma will essentially allow us to hide whether we are choosing our update matrix from the proper distribution or from a more restrictive distribution that potentially causes a cancelation. This lemma holds modulo a prime p, so for these parts of the proof, we will (locally) apply a hybrid argument over the four primes dividing our group order, N. Ultimately, this leads us to a rather elaborate sequence of games, but each individual transition follows somewhat naturally either from a subgroup decision assumption or from an application of the lemma.

Once we arrive at a game with no G_{p_2} components on any of the secret keys and signatures, we show that an attacker has only a negligible chance of producing a forgery that *does have* G_{p_2} parts. Hence we have shown that an attacker has only a negligible chance of producing any forgeries at all.

We now formally define our main sequence of games. We begin with the real security game from Definition 1, which we denote by Game_{Real} (the leakage bound ℓ is implicit here). We next define $\text{Game}_{Real'}$, which is like Game_{Real}, except that the attacker must produce a forgery for a message m^* which is unequal to any queried m modulo p_2. We maintain this additional restriction throughout the rest of the games. To define the additional games, we first define two distributions of n-tuples of elements of G_{p_2}.

Distribution D_{Real} We define distribution D_{Real} as follows. We choose a random element $g_2 \in G_{p_2}$ and three random vectors $\vec{c} = (c_1, \ldots, c_n), \vec{d} = (d_1, \ldots, d_n), \vec{f} = (f_1, \ldots, f_n) \in \mathbb{Z}_N^n$. We output the following three n-tuples of elements in G_{p_2}:

$$g_2^{\vec{c}},\ g_2^{\vec{d}},\ g_2^{\vec{f}}.$$

Note that this is exactly the distribution of the G_{p_2} parts of the secret key SK_0 produced by the key generation algorithm in our signature scheme.

Distribution D_{Alt} We define distribution D_{Alt} as follows. We choose a random element $g_2 \in G_{p_2}$ and three random vectors $\vec{c}, \vec{d}, \vec{f} \in \mathbb{Z}_N^n$ subject to the constraint that these are from a two dimensional subspace. Equivalently, we choose $\vec{c}, \vec{d}$ uniformly at random, and then choose $\vec{f}$ to be a random linear combination of $\vec{c}$ and $\vec{d}$. We output the following three n-tuples of elements in G_{p_2}:

$$g_2^{\vec{c}},\ g_2^{\vec{d}},\ g_2^{\vec{f}}.$$

We let q denote the number of signing queries made by the attacker. We now define the following games:

Game$_i$ In Game$_i$ (for $i \in \{3, \ldots, q+3\}$), the key generation phase happens as in Game$_{Real}$ (in particular, the G_{p_2} parts of SK_0 have distribution D_{Real}). The challenger follows the prescribed signing/update algorithms for the first $i-3$ requested signatures. On the $i-2$ requested signature, the challenger chooses a random update matrix subject to an additional condition. To describe this condition, we let $A_1, \ldots, A_{i-3}$ denote the matrices used in the first $i-3$ updates. We let $A = A_{i-3} \cdots A_1$ denote the product of these update matrices (if $i = 3$, then A denotes the identity matrix). Then, since the G_{p_2} parts of the secret key begin as $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$, the G_{p_2} parts of the secret key after these $i-3$ updates are $g_2^{A\vec{c}}$, $g_2^{A\vec{d}}$, and $g_2^{A\vec{f}}$. The new update matrix A_{i-2} is chosen randomly up to the constraint that the kernel of $A_{i-2}A$ now includes a random vector from the three dimensional subspace spanned by $\vec{c}$, $\vec{d}$, and $\vec{f}$. This means that the challenger will choose a random vector $\vec{w}$ from the span of $\vec{c}, \vec{d}, \vec{f}$ and compute $A\vec{w}$. We let $\vec{w'} = (w'_1, \ldots, w'_n)$ denote the vector $A\vec{w}$. With all but negligible probability, w'_n is invertible modulo N. The challenger will set the vector $\vec{b}_{i-2} = (b_1^{i-2}, \ldots, b_{n-1}^{i-2})$ for the last column of A_{i-2} as follows:

$$b_j^{i-2} := -\frac{w'_j}{w'_n}$$

for each j from 1 to $n-1$. The challenger will choose the vector $\vec{a}_{i-2}$ for the last row of A_{i-2} randomly. After this update is applied, the G_{p_2} parts of the new key will have three exponent vectors in the same 2-dimensional space, i.e. they will be $g_2^{A_{i-2}\cdot A\cdot\vec{c}}$, $g_2^{A_{i-2}A\vec{d}}$, and $g_2^{A_{i-2}A\vec{f}}$. (To see that the vectors $A_{i-2}A\vec{c}, A_{i-2}A\vec{d}$, and $A_{i-2}A\vec{f}$ now span a 2-dimensional space, note that one basis for this space is $A_{i-2}A\vec{v}, A_{i-2}A\vec{t}$, where $\vec{w}, \vec{v}, \vec{t}$ are an alternate basis for the span of $\vec{c}, \vec{d}, \vec{f}$.)

For the $i-1$ requested signature, the challenger chooses the new update matrix A_{i-1} randomly subject to the constraint that the kernel of A_{i-1} now includes a random vector from the two dimensional subspace spanned by $A_{i-2}A\vec{c}$, $A_{i-2}A\vec{d}$, and $A_{i-2}A\vec{f}$. After this update is applied, the G_{p_2} parts of the new key have exponent vectors $A_{i-1}A_{i-2}A\vec{c}$, $A_{i-1}A_{i-2}A\vec{d}$, and $A_{i-1}A_{i-2}A\vec{f}$ which all lie in the same 1-dimensional subspace.

For the i^{th} update, A_i is chosen randomly up to the constraint that the kernel of A_i now includes this 1-dimensional space (in our notation, $A_iA_{i-1}A_{i-2}A\vec{c}$ is equal to the vector of all 0's, and the same holds for $\vec{d}$ and $\vec{f}$). This cancels out the G_{p_2} parts of the secret key, and all subsequently produced signatures will not contain G_{p_2} parts. The remaining update matrices are chosen from the usual distribution specified in the update algorithm.

GameAlt$_i$ In GameAlt$_i$ (for $i \in \{2, \ldots, q+2\}$), the key generation phase differs from Game$_{Real}$ in that the G_{p_2} components of SK_0 have distribution D_{Alt} instead of D_{Real}. In other words, the G_{p_2} parts are set as $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$, where $\vec{c}, \vec{d}$ are chosen randomly and $\vec{f}$ is chosen randomly from the span of $\vec{c}$ and $\vec{d}$. All other aspects of the key generation are the same. The challenger follows the prescribed signing/update algorithms for the first $i-2$ requested signatures. We let $A = A_{i-2} \ldots A_1$ denote the product of the first $i-2$ update matrices (if $i = 2$, we let A denote the identity matrix).

For the $i-1$ requested signature, the challenger chooses the update matrix A_{i-1} randomly subject to the constraint that the kernel of $A_{i-1}A$ now includes a random vector from the two dimensional subspace spanned by $\vec{c}$ and $\vec{d}$. After this update is applied, the G_{p_2} parts of the new key have exponent vectors $A_{i-1}A\vec{c}$, $A_{i-1}A\vec{d}$, and $A_{i-1}A\vec{f}$ which all lie in the same 1-dimensional subspace.

For the i^{th} update, A_i is chosen randomly up to the constraint that the kernel of A_i now includes this 1-dimensional space (in our notation, $A_iA_{i-1}A\vec{c}$ is equal to the vector of all 0's, and the same holds for $\vec{d}$ and $\vec{f}$). This cancels out the G_{p_2} parts of the secret key, and all subsequently produced signatures will not contain G_{p_2} parts.

For the $i+1$ update, A_{i+1} is chosen so that the kernel of $A_{i+1}A_iA_{i-1}A$ now includes a new uniformly random vector. In other words, a random vector $\vec{t}$ is chosen, and A_{i+1} is chosen randomly in the form prescribed by the update algorithm up to the additional constraint that $A_{i+1}A_iA_{i-1}A\vec{t}$ is the all zeros vector. The remaining update matrices are chosen from the usual distribution specified in the update algorithm.

GameAlt$_1$ In GameAlt$_1$, the secret key is initialized to have G_{p_2} components of the form $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, $g_2^{\vec{f}}$, where $\vec{c}, \vec{d}, \vec{f}$ are all in the same 1-dimension subspace. The first update matrix, A_1, is chosen so that $\vec{c}$ is in its kernel (and hence $\vec{d}, \vec{f}$ are as well). The next two update matrices, A_2, A_3 are each chosen so that a new random vector is added to the kernel of the product (so $A_3A_2A_1$ will have rank $n-3$). The remaining update matrices are chosen from the usual distribution specified in the update algorithm.

GameAlt$_0$ In GameAlt$_0$, the secret key is initialized to have no G_{p_2} components. All other aspects of the key generation are the same as Game$_{Real}$. The first three update matrices, A_1, A_2, A_3 are each chosen so that a new random vector is added to the kernel of the product each time. All of the remaining update matrices are chosen according to the usual distribution specified in the update algorithm. Note that none of the produced signatures will have any G_{p_2} parts.

We will prove our scheme is ℓ-leakage resilient in the sense of Definition 1 via a hybrid argument over these games. We first show that an attacker's advantage can change only negligibly when we switch from Game$_{Real}$ to Game$_{Real'}$. We then divide forgeries into two classes: Type I and Type II. We say the attacker has produced a *Type I forgery* if the group elements (σ_1, σ_2) of the (correctly verifying) signature contain no G_{p_2} parts. We define Type II forgeries in a complimentary way: i.e. a verifying signature (σ_1, σ_2) is a *Type II forgery* if at least one of σ_1, σ_2 has a G_{p_2} part. We show that in Game$_{Real'}$, the attacker's chance of producing a Type I forgery is negligible.

We note that Game$_{Real'}$ is exactly the same as Game$_{q+3}$ (the first q updates are normal, and the attacker only asks for q signatures). For i from 2 to $q+2$, we show that the attacker's chance of producing a Type II forgery changes only negligibly when we change from Game$_{i+1}$ to GameAlt$_i$ (the leakage parameter ℓ will play a role here). We will also prove that the attacker's chance of producing a Type II forgery changes only negligibly when we change from GameAlt$_i$ to Game$_i$ (the leakage parameter ℓ will also play a role in these transitions). This allows us traverse the games from Game$_{Real'}$ to GameAlt$_{q+2}$, then to Game$_{q+2}$, then to GameAlt$_{q+1}$, then to Game$_{q+1}$, and so on, until we arrive at GameAlt$_2$. We finally show that the attacker's chance of producing a Type II forgery differs only negligibly between GameAlt$_2$ and GameAlt$_1$ and between GameAlt$_1$ and GameAlt$_0$, and also that the attacker can only produce a Type II forgery with negligible probability in GameAlt$_0$. Since we have taken a polynomial number of steps, this means that the attacker can only produce a Type II forgery with negligible probability in Game$_{Real'}$. Since any forgery must be either a Type I or a Type II forgery, we have then proven security. We execute this proof strategy in the following subsections. The proofs of many of the lemmas will be very similar to each other, but we include them all in full for completeness.

5.1 Transition from Game$_{Real}$ to Game$_{Real'}$

We first show:

Lemma 6. *Under Assumptions 1 and 3, any polynomial time attacker $\mathcal{A}$ has only a negligibly different probability of winning in Game$_{Real}$ versus Game$_{Real'}$.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which attains a non-negligible difference in probability of winning between Game$_{Real}$ and Game$_{Real'}$. We will create a PPT algorithm $\mathcal{B}$ that breaks either Assumption 1 or Assumption 3 with non-negligible advantage. We first note that the terms given to $\mathcal{B}$ in Assumption 1 (namely $g_3, g_4, X_1X_4, Y_1Y_2Y_3, Z_2Z_3, T$) can be used to properly simulate Game$_{Real}$ with $\mathcal{A}$, and this also holds for the terms given to $\mathcal{B}$ in Assumption 3 (namely $g, g_3, g_4, X_1X_2, Y_2Y_3, T$).

To attain its non-negligible difference in success probability, $\mathcal{A}$ must with non-negligible probability produce $m, m^* \in \mathbb{Z}_N$ during this simulation such that $m \neq m^*$ as elements of $\mathbb{Z}_N$, but $m = m^*$ modulo p_2. For each m, m^* produced by $\mathcal{A}$, $\mathcal{B}$ will compute the greatest common divisor of $m - m^*$ and N. If these values are always equal to 1, then $\mathcal{B}$ guesses randomly for the nature of T. However, with non-negligible probability, at least one of the g.c.d.'s will be strictly between 1 and N.

$\mathcal{B}$ then proceeds as follows. It sets $a = \gcd(m - m^*, N)$ and $b = N/a$. First, we consider the case where one of a, b is equal to p_4, and the other is equal to $p_1p_2p_3$. Without loss of generality, we can say that $a = p_4$ and $b = p_1p_2p_3$. In this case, $\mathcal{B}$ will break Assumption 1. It first tests that $a = p_4$ and $b = p_1p_2p_3$ by checking that $g_4^a = 1$, and $(Y_1Y_2Y_3)^b = 1$. It computes $(X_1X_4)^a \in G_{p_1}$, and pairs this with T. If $T \in G_{p_2p_4}$, this will yield the identity. If $T \in G_{p_1p_2p_4}$, it will not. Thus, $\mathcal{B}$ can break Assumption 1 with non-negligible advantage.

In all other cases, $\mathcal{B}$ will break Assumption 3. We consider 2 cases. For case 1), we suppose that p_1 divides one of a, b and p_2 divides the other. Without loss of generality, we can say that p_1 divides a and p_2 divides b. $\mathcal{B}$ can confirm this by checking that $g^a = 1$ and $(X_1X_2)^a \neq 1$. $\mathcal{B}$ can then test whether T has a G_{p_2} component by pairing T with $(X_1X_2)^a$ and seeing if the result is 1 or not.

For case 2), we suppose that p_3 divides one of a, b and p_2 divides the other. Without loss of generality, we can say that p_3 divides a and p_2 divides b. $\mathcal{B}$ can confirm this by checking that $g_3^a = 1$ and $(Y_2Y_3)^a \neq 1$. It can then pair T with $(Y_2Y_3)^a$ to see whether T has a G_{p_2} component or not.

Now, since $\mathcal{A}$ must produce m, m^* such that $\gcd(m - m^*, N) \neq 1, N$ with non-negligible probability, at least one of these cases must occur with non-negligible probability. Hence, we obtain either a $\mathcal{B}$ which breaks Assumption 1 with non-negligible advantage, or a $\mathcal{B}$ which breaks Assumption 3 with non-negligible advantage. □

5.2 Security Against Type I Forgeries in Game$_{Real'}$

We now show:

Lemma 7. *Under Assumption 1, any polynomial time attacker $\mathcal{A}$ has only a negligible chance of producing a Type I forgery in Game$_{Real'}$.*

Proof. We suppose that there exists a polynomial time attacker $\mathcal{A}$ who can produce a Type I forgery with non-negligible probability in $\text{Game}_{Real'}$. We will use $\mathcal{A}$ to create a polynomial time algorithm $\mathcal{B}$ to break Assumption 1. $\mathcal{B}$ is given $g_3, g_4, X_1X_4, Y_1Y_2Y_3, Z_2Z_3, T$. $\mathcal{B}$ chooses α, β randomly from $\mathbb{Z}_N$ and sets the public parameters as:

$$R = g_4,\ gR' = X_1X_4,\ uR'' = (X_1X_4)^{\alpha},\ hR''' = (X_1X_4)^{\beta}.$$

It gives these to $\mathcal{A}$. We note that these are properly distributed because the values of α, β modulo p_4 are uncorrelated from their values modulo p_1 by the Chinese Remainder Theorem.

To initialize the secret key, $\mathcal{B}$ chooses vectors $\vec{r}, \vec{c}, \vec{d}, \vec{f}, \vec{x}, \vec{y}, \vec{z}$ randomly from $\mathbb{Z}_N^n$. It sets:

$$\vec{S}_0 = (Y_1Y_2Y_3)^{\vec{r}}(Z_2Z_3)^{\vec{c}} g_3^{\vec{x}},$$

$$\vec{U}_0 = (Y_1Y_2Y_3)^{\alpha\vec{r}}(Z_2Z_3)^{\vec{d}} g_3^{\vec{y}},$$

$$\vec{H}_0 = (Y_1Y_2Y_3)^{\beta\vec{r}}(Z_2Z_3)^{\vec{f}} g_3^{\vec{z}}.$$

We note that this is properly distributed. The simulator can now answer all signing and leakage queries by choosing random updates to this secret key and computing the requested leakage as a function of the secret key and update matrix. It can easily produce the requested signatures because it knows the secret keys.

With non-negligible probability, $\mathcal{A}$ produces a forgery $(m^*, \sigma_1, \sigma_2)$ which passes the verification algorithm. When this happens, $\mathcal{B}$ tests whether $e(T, \sigma_1) = 1$. If this is true, $\mathcal{B}$ guesses that $T \in G_{p_2p_4}$. Otherwise, $\mathcal{B}$ guesses randomly. When $\mathcal{A}$ has produced a Type I forgery and $T \in G_{p_2p_4}$, $e(T, \sigma_1) = 1$ will always hold (since a forgery that verifies correctly cannot have any G_{p_4} components). If $T \in G_{p_1p_2p_4}$, then $e(T, \sigma_1) = 1$ will only hold when σ_1 has no G_{p_1} part. This is impossible for a signature that verifies correctly, since the verification algorithm checks that σ_1 has no G_{p_4} parts and also that $e(\sigma_1, gR') \neq 1$. This means that $\mathcal{B}$ achieves a non-negligible advantage, hence breaking Assumption 1. □

5.3 Security Against Type II Forgeries in GameAlt_0

We now show:

Lemma 8. *Under Assumption 2, any polynomial time attacker $\mathcal{A}$ has only a negligible chance of producing a Type II forgery in $GameAlt_0$.*

Proof. We suppose there exists a polynomial time attacker $\mathcal{A}$ who can produce a Type II forgery with non-negligible probability in GameAlt_0. We will use $\mathcal{A}$ to create a polynomial time algorithm $\mathcal{B}$ to break Assumption 2. $\mathcal{B}$ is given g, g_3, g_4, T. $\mathcal{B}$ chooses $\alpha, \beta, \delta, \gamma, \psi$ randomly from $\mathbb{Z}_N$ and sets the public parameters as:

$$R = g_4,\ gR' = gg_4^{\delta},\ uR'' = g^{\alpha}g_4^{\gamma},\ hR''' = g^{\beta}g_4^{\psi}.$$

It gives these to $\mathcal{A}$.

To initialize the secret key, $\mathcal{B}$ chooses vectors $\vec{r}, \vec{x}, \vec{y}, \vec{z}$ randomly from $\mathbb{Z}_N^n$ and sets:

$$\vec{S}_0 = g^{\vec{r}} g_3^{\vec{x}},\ \vec{U}_0 = g^{\alpha\vec{r}} g_3^{\vec{y}},\ \vec{H}_0 = g^{\beta\vec{r}} g_3^{\vec{z}}.$$

This is properly distributed for GameAlt_0. The simulator can now answer all signing and leakage queries by choosing updates to this secret key distributed as specified for GameAlt_0 and computing the requested leakage as a function of the secret key and update matrix. It can produce the requested signatures because it always knows the secret key.

With non-negligible probability, $\mathcal{A}$ produces a forgery $(m^*, \sigma_1, \sigma_2)$ that passes the verification algorithm. When this happens, $\mathcal{B}$ tests whether $e(\sigma_1, T) = e(\sigma_2, T^{\alpha m^* + \beta})$. When this test fails, $\mathcal{B}$ guesses that $T \in G_{p_1 p_2}$. Otherwise, $\mathcal{B}$ guesses randomly. Observe that if this test fails, T must have a G_{p_2} part. When $T \in G_{p_1}$, $e(\sigma_1, T) = e(\sigma_2, T^{\alpha m^* + \beta})$ will hold for any signature that verifies correctly. We note that the values of α, β modulo p_2 are information theoretically hidden from $\mathcal{A}$, so when $\mathcal{A}$ produces a Type II forgery and $T \in G_{p_1 p_2}$, there is only a negligible chance of this test passing. Hence $\mathcal{B}$ has a non-negligible advantage in breaking Assumption 2. □

5.4 Transition from Game$_{i+1}$ to GameAlt$_i$

We now prove:

Lemma 9. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s probability of producing a Type II forgery between Game$_{i+1}$ and GameAlt$_i$ is negligible as long as $\ell \leq \frac{1}{3}(\log(p_2) - 2\delta)$, for each i from 2 to $q+2$. Here, $\delta > 0$ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Proof. We suppose $\mathcal{A}$ is a PPT attacker which achieves a non-negligible difference in probability of producing a Type II forgery between Game$_{i+1}$ and GameAlt$_i$ (for some fixed i). We will create a PPT algorithm $\mathcal{B}$ which achieves non-negligible advantage against Assumption 3.

$\mathcal{B}$ is given $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It will simulate either Game$_{i+1}$ or GameAlt$_i$ with $\mathcal{A}$, depending on the value of T. We will then show that with all but negligible probability, $\mathcal{B}$ can determine when $\mathcal{A}$ is producing a Type II forgery. Thus, the non-negligible difference in $\mathcal{A}$'s probability of producing a Type II forgery will allow $\mathcal{B}$ to achieve non-negligible advantage against Assumption 3.

$\mathcal{B}$ chooses random vectors $\vec{r}, \vec{t}, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and random values $\alpha, \beta, f_1, f_2, \delta, \gamma, \psi \in \mathbb{Z}_N$. It sets the public parameters as:

$$R := g_4,\ gR' := gg_4^{\delta},\ uR'' := g^{\alpha} g_4^{\gamma},\ hR''' := g^{\beta} g_4^{\psi}.$$

It initializes the secret key as:

$$\vec{S}_0 = g^{\vec{r}} T^{\vec{t}} (Y_2Y_3)^{\vec{c}} g_3^{\vec{x}},$$
$$\vec{U}_0 = g^{\alpha\vec{r}} T^{\alpha\vec{t}} (Y_2Y_3)^{\vec{d}} g_3^{\vec{y}},$$
$$\vec{H}_0 = g^{\beta\vec{r}} T^{\beta\vec{t}} (Y_2Y_3)^{f_1\vec{c}+f_2\vec{d}} g_3^{\vec{z}}.$$

We note that the G_{p_1} parts here are properly distributed. To see this, note that if we let g^{τ} denote the G_{p_1} part of T, then the exponents vectors for the G_{p_1} parts are $\vec{r} + \tau\vec{t}$, $\alpha(\vec{r} + \tau\vec{t})$ and $\beta(\vec{r} + \tau\vec{t})$. This is properly distributed because $\vec{r} + \tau\vec{t}$ is a uniformly random vector in $\mathbb{Z}_N^n$. The G_{p_3} parts are also properly distributed because the vectors $\vec{x}, \vec{y}, \vec{z}$ are uniformly random.

Now, if $T \in G_{p_1p_3}$, then the G_{p_2} parts here are distributed according to distribution D_{Alt}. If $T \in G_{p_1p_2p_3}$, then the G_{p_2} parts are distributed according to distribution D_{Real}.

For the first $i-2$ requested signatures, the simulator chooses random update matrices $A_1, \ldots, A_{i-2}$ according to the distribution prescribed in the update algorithm. It provides $\mathcal{A}$ with the requested signatures and leakage values. We let $A = A_{i-2} \cdots A_1$ denote the product of all the update matrices applied so far.

For the $i-1$ requested signature, $\mathcal{B}$ chooses an update matrix A_{i-1} whose rows are orthogonal to $A\vec{w}$, where $\vec{w}$ is randomly chosen from the span of $\vec{c}$ and $\vec{d}$. With respect to the entries

$b_1^{i-1}, \dots, b_{n-1}^{i-1}, a_1^{i-1}, \dots, a_{n-1}^{i-1}$ of A_{i-1}, this means the following. We let $\vec{w}' = (w'_1, \dots, w'_n)$ denote the vector $A\vec{w}$. We will choose

$$b_j^{i-1} = -\left(\frac{w'_j}{w'_n}\right)$$

for each j from 1 to $n-1$ (with all but negligible probability, w'_n is invertible modulo N), and we will choose the a_j^{i-1} values randomly. This precisely ensures that $\vec{w}$ is in the kernel of $A_{i-1}A$. We let $\vec{v}$ denote a random vector such that the span of $\vec{v}, \vec{w}$ is equal to the span of $\vec{c}, \vec{d}$.

For the i^{th} requested signature, $\mathcal{B}$ chooses an update matrix A_i whose rows are orthogonal to $A_{i-1}A\vec{v}$: this will cancel out the Y_2Y_3 terms. With respect to the entries $b_1^i, \dots, b_{n-1}^i, a_1^i, \dots, a_{n-1}^i$ of A_i, this means the following. We let $\vec{v}' = (v'_1, \dots, v'_n)$ denote the vector $A_{i-1}A\vec{v}$. We will choose

$$b_j^{i} = -\left(\frac{v'_j}{v'_n}\right)$$

for each j from 1 to $n-1$, and we will choose the a_j^i values randomly. This precisely ensures that $\vec{v}$ is in the kernel of $A_iA_{i-1}A$.

For the $i+1$ requested signature, $\mathcal{B}$ chooses an update matrix whose A_{i+1} whose rows are orthogonal to $A_iA_{i-1}A\vec{t}$: this will cancel out the T terms. With respect to the entries $b_1^{i+1}, \dots, b_{n-1}^{i+1}$, $a_1^{i+1}, \dots, a_{n-1}^{i+1}$ of A_{i+1}, this means the following. We let $\vec{t}' = (t'_1, \dots, t'_n)$ denote the vector $A_iA_{i-1}A\vec{t}$. We will choose

$$b_j = -\left(\frac{t'_j}{t'_n}\right)$$

for each j from 1 to $n-1$, and we will choose the a_j^{i+1} values randomly. This precisely ensures that $\vec{t}$ is in the kernel of $A_{i+1}A_iA_{i-1}A$.

$\mathcal{B}$ responds to the remaining signature requests by choosing random update matrices according to the distribution specified in the update algorithm. We note that $\mathcal{B}$ knows all of the update matrices and secret keys throughout the simulation, and so can easily provide the requested leakage and signatures to $\mathcal{A}$. When $T \in G_{p_1p_3}$, $\mathcal{B}$ has properly simulated GameAlt$_i$. To see this, first observe that the Y_2 terms are the only G_{p_2} parts of the secret key, and these are canceled out in the i^{th} update, as required in the specification of GameAlt$_i$. The $i+1$ update is chosen to include a new vector $\vec{t}$ in the kernel, and this vector is uniformly random because $\vec{r}+\tau\vec{t}$ reveals no information about $\vec{t}$ modulo p_1 since $\vec{r}$ is uniformly random, and no information about $\vec{t}$ modulo p_3 is previously revealed because $\vec{x}, \vec{y}, \vec{z}$ are uniformly random. Thus, this is a proper simulation of GameAlt$_i$.

When $T \in G_{p_1p_2p_3}$, $\mathcal{B}$ has properly simulated Game$_{i+1}$. To see this, note that the G_{p_2} parts of the key originally have exponent vectors which are randomly distributed in the three dimensional subspace spanned by $\vec{c}$, $\vec{d}$, and $\vec{t}$. Here, we have chosen the $i-1$ update to cancel out one random dimension of this subspace, the i^{th} update to cancel out another random dimension, and the $i+1$ update to cancel the final dimension. This precisely matches the specification of Game$_{i+1}$.

When $\mathcal{A}$ produces a forgery (σ_1, σ_2) on m^* (that verifies correctly), $\mathcal{B}$ must determine whether it is a Type I or Type II forgery. It tests whether:

$$e(\sigma_1, X_1X_2) \stackrel{?}{=} e(\sigma_2, (X_1X_2)^{\alpha m^* + \beta}).$$

If this equality holds, $\mathcal{B}$ will guess that $\mathcal{A}$ has produced a Type I forgery. If the equality fails, then $\mathcal{B}$ knows that $\mathcal{A}$ has produced a Type II forgery (note that this equality can only fail for a forgery that properly verifies when there is some G_{p_2} part present in σ_1 and/or σ_2).

Finally, we must argue that $\mathcal{A}$ can only produce a Type II forgery which satisfies the equality above with negligible probability. This means that $\mathcal{B}$ will have only negligible error in determining the forgery type produced by $\mathcal{A}$, and hence it can use the output of $\mathcal{A}$ to achieve non-negligible advantage against Assumption 3. In order to produce a Type II forgery that $\mathcal{B}$ misclassifies as a Type I forgery, $\mathcal{A}$ must produce G_{p_2} parts for σ_1 and σ_2 of the form g_2^s, $g_2^{s(\alpha m^*+\beta)}$, where g_2 is a generator of G_{p_2} and s is arbitrary. In other words, $\mathcal{A}$ must be able to implicitly determine the value $\alpha m^* + \beta$ modulo p_2.

Now, if $T \in G_{p_1p_3}$, then the initial secret key reveals *no information* about the values of α and β modulo p_2: so these remain information-theoretically hidden from $\mathcal{A}$ throughout the entire game. Thus, $\mathcal{A}$ has only a negligible chance of determining $\alpha m^* + \beta$ modulo p_2 correctly. When $T \in G_{p_1p_2p_3}$, we will first argue that the values of α and β modulo p_2 are information-theoretically hidden from $\mathcal{A}$ until A_{i-1} is chosen (i.e. for the first $i-2$ updates).

We let g_2^τ denote the G_{p_2} part of T, and define y modulo p_2 by $g_2^y = Y_2$. Then the initial G_{p_2} parts of the secret key are $g_2^{\tau\vec{t}+y\vec{c}}$, $g_2^{\alpha\tau\vec{t}+y\vec{d}}$, and $g_2^{\beta\tau\vec{t}+f_1y\vec{c}+f_2y\vec{d}}$. These three exponent vectors are distributed as uniformly random vectors modulo p_2, and reveal no information about β, α. More specifically, we note that with all but negligible probability, the three exponent vectors will be linearly independent and almost all choices of α, β, f_1, f_2 will lead to the same number of (equally likely) solutions for $y\vec{c}$, $y\vec{d}$, and $\tau\vec{t}$. Thus, with all but negligible probability, no information about the values of α, β modulo p_2 is revealed by these exponent vectors. This remains true (with all but negligible probability) as we choose update matrices $A_1, \ldots, A_{i-2}$ randomly from the distribution specified by the update algorithm.

Now, when we choose A_{i-1}, A_i, and A_{i+1} to progressively cancel out the span of $\vec{c}, \vec{d}, \vec{t}$, we will leak information about α, β modulo p_2. However, after A_{i+1} is applied, the values of α and β modulo p_2 no longer appear, since there are no G_{p_2} terms in the secret key from this point on. Also, all of the update matrices are independent of α, β. Thus, the attacker gets only three chances to obtain leakage on the values of α, β modulo p_2. After update A_{i-1} is applied, the attacker will also receive a signature on m_{i-1} (the $i-1$ requested message) whose G_{p_2} parts still do not reveal any information about the values of α, β modulo p_2. This holds because the signature only involves the first row of A_{i-1}, and this row alone is still properly distributed (its first entry is 1 and its last entry is uniformly random). To see this, note that the first and last entries of $\vec{w'} = A\vec{w}$ are random with all but negligible probability when $\vec{w}$ is chosen randomly from the span of $\vec{c}, \vec{d}$. This is because the first and last rows of A are nonzero and independent of $\vec{c}, \vec{d}$. However, when the i^{th} signature is produced for m_i, the attacker will receive a signature with G_{p_2} parts of the form $g_2^{s(\alpha m_i+\beta)}, g_2^s$ for some s modulo p_2. This information-theoretically reveals $\alpha m_i + \beta$ modulo p_2. Since $\alpha m + \beta$ is a pairwise independent function of m modulo p_2, this means that the attacker still has no information about $\alpha m + \beta$ for any $m \neq m_i$ modulo p_2.

We let X denote the random variable $\alpha || \beta$ modulo p_2 (the $||$ symbol here denotes concatenation). This is a random variable with min-entropy $2\log(p_2)$. The information the attacker learns about X (information-theoretically) can be expressed as $F(X)$ for a single function F which produces $3\ell + \log(p_2)$ bits (3ℓ bits learned from three leakage queries and $\log(p_2)$ bits learned from $\alpha m_i + \beta$ modulo p_2). Thus, for $\ell \leq \frac{1}{3}(\log(p_2) - 2\delta)$, by Lemma 3, the min-entropy of X conditioned on $F(X)$ will be at least δ with probability $1 - 2^{-\delta}$ (which is all but negligible probability). In this case, the probability of an attacker determining $\alpha m^* + \beta$ modulo p_2 correctly for some $m^* \neq m_i$ modulo p_2 is at most $2^{-\delta}$, which is negligible (note that $\alpha m^* + \beta$ and $\alpha m_i + \beta$ together would fully determine α, β since m^*, m_i are known). Recall that we have restricted the attacker to producing forgeries for m^* which are not equal to the queried messages modulo p_2. This completes the proof that $\mathcal{B}$ will incur only negligible error in determining the forgery type of $\mathcal{A}$, and hence will achieve non-negligible advantage against Assumption 3. $\square$

5.5 Transition from GameAlt$_i$ to Game$_i$

To prove that $\mathcal{A}$'s chance of producing a Type II forgery changes only negligibly between GameAlt$_i$ and Game$_i$, we need to introduce a few additional games.

GameAlt$'_i$ This game is like GameAlt$_i$, except the update matrix for the $i+1$ update is now chosen from the distribution specified in the update algorithm. (Recall that in GameAlt$_i$, the $i+1$ update matrix was chosen to include a new vector in the kernel of the matrix product.)

GameAlt$''_i$ This game is like GameAlt$'_i$, except that the matrix for the $i-2$ update is now chosen to include a new random vector in the kernel of the matrix product (i.e. the product $A_{i-2}A$, where A is the product of all the previous update matrices). We note that for $i = 3$, this is the same as GameAlt$'_i$, since the first update matrix is rank $n-1$.

We will first show that $\mathcal{A}$'s chance of producing a Type II forgery changes only negligibly between Game$_i$ and GameAlt$''_i$. We will then show that GameAlt$''_i$ is indistinguishable from GameAlt$'_i$ in $\mathcal{A}$'s view, and finally that GameAlt$'_i$ and GameAlt$_i$ are indistinguishable in $\mathcal{A}$'s view.

Lemma 10. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s probability of producing a Type II forgery between Game$_i$ and GameAlt$''_i$ is negligible as long as $\ell \leq \log(p_2) - 2\delta$, for each i from 3 to $q+2$. Here, $\delta > 0$ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Proof. We suppose there exists a PPT algorithm $\mathcal{A}$ which achieves a non-negligible difference in probability of producing a Type II forgery between Game$_i$ and GameAlt$''_i$ (for some fixed i). We will create a PPT algorithm $\mathcal{B}$ which achieves non-negligible advantage against Assumption 3.

$\mathcal{B}$ is given $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It will simulate either Game$_i$ or GameAlt$''_i$ with $\mathcal{A}$, depending on the value of T. We will then show that with all but negligible probability, $\mathcal{B}$ can determine when $\mathcal{A}$ is producing a Type II forgery. Thus, the non-negligible difference in $\mathcal{A}$'s probability of producing a Type II forgery will allow $\mathcal{B}$ to achieve non-negligible advantage against Assumption 3.

As in the proof of Lemma 9, $\mathcal{B}$ chooses random vectors $\vec{r}, \vec{t}, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and random values $\alpha, \beta, f_1, f_2, \delta, \gamma, \psi \in \mathbb{Z}_N$. It sets the public parameters as:

$$R := g_4,\ gR' := gg_4^{\delta},\ uR'' := g^{\alpha}g_4^{\gamma},\ hR''' := g^{\beta}g_4^{\psi}.$$

It initializes the secret key as:

$$\vec{S}_0 = g^{\vec{r}}T^{\vec{t}}(Y_2Y_3)^{\vec{c}}g_3^{\vec{x}},$$

$$\vec{U}_0 = g^{\alpha\vec{r}}T^{\alpha\vec{t}}(Y_2Y_3)^{\vec{d}}g_3^{\vec{y}},$$

$$\vec{H}_0 = g^{\beta\vec{r}}T^{\beta\vec{t}}(Y_2Y_3)^{f_1\vec{c}+f_2\vec{d}}g_3^{\vec{z}}.$$

As noted in the proof of Lemma 9, the G_{p_1} and G_{p_3} parts here are properly distributed.

If $T \in G_{p_1p_3}$, then the G_{p_2} parts here are distributed according to distribution D_{Alt}. If $T \in G_{p_1p_2p_3}$, then the G_{p_2} parts are distributed according to distribution D_{Real}.

For the first $i-3$ requested signatures, the simulator chooses random update matrices $A_1, \ldots, A_{i-3}$ according to the distribution prescribed in the update algorithm. It provides $\mathcal{A}$ with the requested signatures and leakage values. We let $A = A_{i-3}\cdots A_1$ denote the product of all the update matrices applied so far (if $i = 3$, then A is the identity matrix).

For the $i-2$ requested signature, $\mathcal{B}$ chooses an update matrix A_{i-2} whose rows are orthogonal to $A\vec{t}$. With respect to the entries $b_1^{i-2}, \ldots, b_{n-1}^{i-2}, a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ of A_{i-2}, this means the following. We let $\vec{t}' = (t'_1, \ldots, t'_n)$ denote the vector $A\vec{t}$. We will choose

$$b_j^{i-2} = -\left(\frac{t'_j}{t'_n}\right)$$

for each j from 1 to $n-1$ (with all but negligible probability, t_n is invertible modulo N), and we will choose the a_j^{i-2} values randomly. This precisely ensures that $\vec{t}$ is in the kernel of $A_{i-2}A$. We note that when $T \in G_{p_1p_2p_3}$, this is a proper simulation of the $i-2$ update in Game$_i$, and when $T \in G_{p_1p_3}$, this is a proper simulation of the $i-2$ update in Game$''_i$, since $\vec{t}$ is a uniformly random vector (note that no information about $\vec{t}$ modulo p_1 is revealed by $\vec{r}+\vec{t}$, since $\vec{r}$ is also random, and no information about $\vec{t}$ is revealed modulo p_3 because $\vec{x}, \vec{y}, \vec{z}$ are random).

For the $i-1$ requested signature, $\mathcal{B}$ chooses an update matrix A_{i-1} whose rows are orthogonal to $A_{i-2}A\vec{w}$, where $\vec{w}$ is randomly chosen from the span of $\vec{c}$ and $\vec{d}$. With respect to the entries $b_1^{i-1}, \ldots, b_{n-1}^{i-1}, a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ of A_{i-1}, this means the following. We let $\vec{w}' = (w'_1, \ldots, w'_n)$ denote the vector $A_{i-2}A\vec{w}$. We will choose

$$b_j^{i-1} = -\left(\frac{w'_j}{w'_n}\right)$$

for each j from 1 to $n-1$ (with all but negligible probability, w_n is invertible modulo N), and we will choose the a_j^{i-1} values randomly. This precisely ensures that $\vec{w}$ is in the kernel of $A_{i-1}A_{i-2}A$. We let $\vec{v}$ denote a random vector such that the span of $\vec{v}, \vec{w}$ is equal to the span of $\vec{c}, \vec{d}$.

For the i^{th} requested signature, $\mathcal{B}$ chooses an update matrix A_i whose rows are orthogonal to $A_{i-1}A_{i-2}A\vec{v}$: this will cancel out the Y_2Y_3 terms (and there will be no G_{p_2} remaining). With respect to the entries $b_1^i, \ldots, b_{n-1}^i, a_1^i, \ldots, a_{n-1}^i$ of A_i, this means the following. We let $\vec{v}' = (v'_1, \ldots, v'_n)$ denote the vector $A_{i-1}A_{i-2}A\vec{v}$. We will choose

$$b_j^i = -\left(\frac{v'_j}{v'_n}\right)$$

for each j from 1 to $n-1$, and we will choose the a_j^i values randomly. This precisely ensures that $\vec{v}$ is in the kernel of $A_iA_{i-1}A_{i-2}A$. The remaining updates are chosen from the distribution prescribed by the update algorithm. Since $\mathcal{B}$ knows all of the update matrices and secret keys, it can easily produce the signatures and leakage requested by $\mathcal{A}$.

We note that the $i-1$ and i updates are proper simulations of Game$_i$ and GameAlt$''_i$. Hence, when $T \in G_{p_1p_2p_3}$, $\mathcal{B}$ has properly simulated Game$_i$. When $T \in G_{p_1p_3}$, $\mathcal{B}$ has properly simulated GameAlt$''_i$. We must now argue that $\mathcal{B}$ can accurately detect the forgery type produced by $\mathcal{A}$, with only negligible error. We proceed similarly to the proof of Lemma 9.

When $\mathcal{A}$ produces a (correctly verifying) forgery (σ_1, σ_2) on m^*, $\mathcal{B}$ determines whether it is a Type I or Type II forgery by testing:

$$e(\sigma_1, X_1X_2) \stackrel{?}{=} e(\sigma_2, (X_1X_2)^{\alpha m^* + \beta}).$$

If this equality holds, $\mathcal{B}$ will guess that $\mathcal{A}$ has produced a Type I forgery. If this equality fails, then $\mathcal{B}$ knows that $\mathcal{A}$ has produced a Type II forgery (note that this equality can only fail for a forgery that properly verifies when there is some G_{p_2} part present in σ_1 and/or σ_2).

We again argue that $\mathcal{A}$ can only produce a Type II forgery which satisfies the equality above with negligible probability. In order to produce a Type II forgery that $\mathcal{B}$ misclassifies as a Type

I forgery, $\mathcal{A}$ must produce G_{p_2} parts for σ_1 and σ_2 of the form g_2^s, $g_2^{s(\alpha m^*+\beta)}$, where g_2 is a generator of G_{p_2} and s is arbitrary. In other words, $\mathcal{A}$ must be able to implicitly determine the value $\alpha m^* + \beta$ modulo p_2.

If $T \in G_{p_1p_3}$, the initial secret key reveals *no information* about the values of α and β modulo p_2: so these remain information-theoretically hidden from $\mathcal{A}$ throughout the entire game. Thus, $\mathcal{A}$ has only a negligible chance of determining $\alpha m^* + \beta$ modulo p_2 correctly. When $T \in G_{p_1p_2p_3}$, we first note that the values of α and β modulo p_2 are information-theoretically hidden from $\mathcal{A}$ until A_{i-2} is chosen (i.e. for the first $i-3$ updates). This holds for the same reasons noted in the proof of Lemma 9.

Now, for the $i-2$ update, the choice of A_{i-2} involves the vector $\vec{t}$, giving the attacker an opportunity to obtain some limited information about the values of α, β modulo p_2 from the leakage on A_{i-2} and the current secret key. (We note that A_{i-2} by itself is independent of α, β, but when this is considered in combination with the current secret key, some information about α, β modulo p_2 is revealed.) This is in fact the attacker's *only* opportunity to learn any information about α, β modulo p_2, since they will be canceled out of the secret key once the update A_{i-2} is applied (in particular, none of the given signatures reveal any information about α, β modulo p_2). We also note that *all* of the update matrices are chosen independently of α, β.

We again let X denote the random variable $\alpha||\beta$ modulo p_2. This has min-entropy $2\log(p_2)$. The information the attacker learns about X can be expressed as $F(X)$ for a single function F which produces ℓ bits (ℓ bits learned from a single leakage query). Thus, for $\ell \leq \log(p_2) - 2\delta$, by Lemma 3, the min-entropy of X conditioned on $F(X)$ will be at least $\log(p_2) + \delta$ with probability $1 - 2^{-\delta}$ (which is all but negligible probability). In this case, the probability of an attacker determining $\alpha m^* + \beta$ modulo p_2 correctly for some m^* is at most $2^{-\delta}$, which is negligible. To see this, note that the min-entropy of X conditioned on $m^*, \alpha m^* + \beta$ is $\log(p_2)$. Thus, if an attacker seeing only $F(X)$ could produce $\alpha m^* + \beta, m^*$ with probability $> 2^{-\delta}$, it could predict the value of X with probability $> 2^{-\delta - \log(p_2)}$, contradicting that X conditioned on $F(X)$ has min-entropy at least $\log(p_2) + \delta$. This completes the proof that $\mathcal{B}$ will incur only negligible error in determining the forgery type of $\mathcal{A}$, and hence will achieve non-negligible advantage against Assumption 3. □

To show that GameAlt$''_i$ is indistinguishable from GameAlt$'_i$ in $\mathcal{A}$'s view, we will use the following lemma from [12]:

Lemma 11. *Let $m, k, d \in \mathbb{N}$, $m \geq k \geq 2d$, and let p be a prime. Let X be a uniformly random matrix in $\mathbb{Z}_p^{m \times k}$, let T be a uniformly random matrix of rank d in $\mathbb{Z}_p^{k \times d}$, and let Y be a uniformly random matrix $\mathbb{Z}_p^{m \times d}$. Let $F : \mathbb{Z}_p^{m \times d} \to W$ be some function. Then,*

$$\mathit{dist}\left((X, F(X \cdot T)), (X, F(Y))\right) \leq \epsilon,$$

as long as

$$|W| \leq 4 \cdot (1 - 1/p) \cdot p^{k-(2d-1)} \cdot \epsilon^2,$$

where $\mathit{dist}(Z_1, Z_2)$ denotes the statistical distance between random variables Z_1 and Z_2.

For convenience, we also state the following immediate corollary [37]:

Corollary 12. *Let $m \in \mathbb{N}$, $m \geq 3$, and let p be a prime. Let $\vec{\delta}, \vec{\tau}$ be uniformly random vectors in $\mathbb{Z}_p^m$, and let $\vec{\tau'}$ be chosen uniformly at random from the set of vectors which are orthogonal to $\vec{\delta}$ under the dot product modulo p. Let $F : \mathbb{Z}_p^m \to W$ be some function. Then:*

$$\mathit{dist}\left((\vec{\delta}, F(\vec{\tau})), (\vec{\delta}, F(\vec{\tau'}))\right) \leq \epsilon,$$

as long as

$$|W| \leq 4 \cdot (1 - 1/p) \cdot p^{m-2} \cdot \epsilon^2$$

holds.

Proof. We apply Lemma 11 with $d = 1$ and $k = m - 1$. Y corresponds to $\vec{\tau}$, and X corresponds to a basis for the orthogonal space of $\vec{\delta}$. Then $\vec{\tau'}$ is distributed as $X \cdot T$, where T is a uniformly random vector in $\mathbb{Z}_p^{k \times 1}$. We note that X is determined by $\vec{\delta}$, and is properly distributed for use in Lemma 11. We have:

$$dist\left((\vec{\delta}, F(\vec{\tau})), (\vec{\delta}, F(\vec{\tau'}))\right) \leq dist\left((X, F(X \cdot T)), (X, F(Y))\right) \leq \epsilon.$$

□

We will also need the following linear algebraic lemma:

Lemma 13. *Let p be a prime, and let C be a matrix of rank $n - c$ over $\mathbb{Z}_p$, for $c < n$. We let $\{\vec{\gamma}_1, \ldots, \vec{\gamma}_c \in \mathbb{Z}_p^n\}$ denote a basis for its left nullspace (i.e. the space of vectors orthogonal to the columns of C). We let $C_1, \ldots, C_n$ denote the rows of C, and we suppose that C_n has at least one non-zero entry. Then, choosing a random vector $\vec{t}$ and setting $b_1, \ldots, b_{n-1}$ as $b_j = -\frac{C_j \cdot \vec{t}}{C_n \cdot \vec{t}}$ yields the same distribution (up to a negligible difference) as choosing a random values for $b_1, \ldots, b_{n-1}$ modulo p up to the constraint that the vector $(b_1, \ldots, b_{n-1}, -1)$ is orthogonal to all of $\vec{\gamma}_1, \ldots, \vec{\gamma}_c$ (i.e. is in the left nullspace of C). (We ignore the negligible event that $C_n \cdot \vec{t} = 0$.)*

Proof. We note that the vector $(b_1, \ldots, b_{n-1}, -1)$ is orthogonal to all of $\vec{\gamma}_1, \ldots, \vec{\gamma}_c$ if and only if it is in the column space of C. $C\vec{t} = (C_1 \cdot \vec{t}, \ldots, C_n \cdot \vec{t})$ is distributed as a random vector in the column space of C. When $C_n \cdot \vec{t} \neq 0$ (which happens with all but negligible probability when $\vec{t}$ is randomly chosen and C_n is non-zero), we can rescale this vector as

$$\left(-\frac{C_1 \cdot \vec{t}}{C_n \cdot \vec{t}}, \ldots, -\frac{C_{n-1}\vec{t}}{C_n \cdot \vec{t}}, -1\right).$$

Thus, choosing $b_1, \ldots, b_{n-1}$ such that $b_j = -\frac{C_j \cdot \vec{t}}{C_n \cdot \vec{t}}$ yields the same distribution (excepting the negligible event that $C_n\vec{t} = 0$) as choosing $(b_1, \ldots, b_{n-1}, -1)$ randomly up to the constraint that it is orthogonal to all of $\vec{\gamma}_1, \ldots, \vec{\gamma}_c$. □

Applying this lemma, we obtain the following properties of our update matrices modulo p_i for each prime p_i dividing N:

Corollary 14. *Let $k \in \mathbb{N}$ be polynomial in the security parameter λ and let p be a prime. Suppose that $A_1, \ldots, A_k$ are randomly chosen $n \times n$ update matrices (according to the distribution prescribed by the update algorithm). We consider these matrices modulo p. Let $\vec{a}_k = (a_1^k, \ldots, a_{n-1}^k)$ denote the values (mod p) used in the last row of A_k. Let $\vec{a}_{k+1}, \vec{b}_{k+1} \in \mathbb{Z}_N^{n-1}$ denote the entries (mod p) used to form A_{k+1}. Then, with all but negligible probability over the choice of $A_1, \ldots, A_k$, we have that choosing A_{k+1} modulo p so that $A_{k+1} \cdots A_1$ includes a new random vector $\vec{t}$ in its kernel modulo p is equivalent (up to a negligible difference) to choosing $\vec{a}_{k+1}$ uniformly at random and $\vec{b}_{k+1}$ at random up to the constraint that $\vec{b}_{k+1} \cdot \vec{a}_k = -1$ modulo p.*

Proof. We define the matrix A by $A = A_k \cdots A_1$. With all but negligible probability, A is a rank $n - 1$ matrix. To see this, note that rank $A_i \cdots A_1$ is less than rank $A_{i-1} \cdots A_1$ for each i if an only if the (1-dimensional) kernel of A_i is contained in the column space of $A_{i-1} \cdots A_1$. If we let $\vec{a}_i$, $\vec{b}_i \in \mathbb{Z}_p^{n-1}$ denote the values used in the last row and column of A_i respectively, then the kernel of A_i is spanned by the length n vector formed by concatenating $\vec{b}_i$ with a -1 as the n^{th} entry. This is a random 1-dimensional space when $\vec{b}_i$ is chosen uniformly at random, and so the probability that it will be contained in the column space of $A_{i-1} \cdots A_1$ is negligible. Since this holds for each i and we assume that k is polynomial, we may conclude that A is a rank $n-1$ matrix with all but negligible probability.

Because the rank of A is $n-1$, the column space of A is equal to the column space of A_k. This is an $(n-1)$-dimensional space, consisting of all vectors which are orthogonal to the vector $\vec{\gamma} := \left(a_1^k, \ldots, a_{n-1}^k, -1\right)$. Now, we consider choosing A_{k+1} so that its rows are all orthogonal to $A\vec{t}$ for a random vector $\vec{t}$. We let $\vec{t'} = (t'_1, \ldots, t'_n)$ denote the vector $A\vec{t}$. Then, (ignoring the negligible probability event that $t'_n = 0$), choosing A_{k+1} so that $A_{k+1}\vec{t'}$ is the all zeros vector is equivalent to choosing $\vec{a}_{k+1}$ uniformly at random and setting the entries of $\vec{b}_{k+1}$ as $b_j^{k+1} = -\frac{t'_j}{t'_n}$. By Lemma 13, this is equivalent to choosing $\vec{b}_{k+1}$ randomly up to the constraint that $\vec{b}_{k+1} \cdot \vec{a}_k = -1$ modulo p. □

Corollary 15. *Let $k \in \mathbb{N}$ be polynomial in the security parameter λ, and let p be a prime. Suppose that $A_1, \ldots, A_{k-2}$ are randomly chosen $n \times n$ update matrices (according to the distribution prescribed by the update algorithm). We consider these matrices modulo p. Let $\vec{a}_i = (a_1^i, \ldots, a_{n-1}^i)$ denote the values modulo p used in the last row of A_i for each i, and let $\vec{b}_i$ denote the values modulo p used in the last column. We let $A = A_{k-2} \cdots A_1$. We suppose that A_{k-1} is chosen modulo p so that $A_{k-1}A\vec{t}$ is the all zeros vector for a randomly chosen vector $\vec{t}$ modulo p, and A_k is chosen so that $A_kA_{k-1}A\vec{v}$ is the all zeros vector for a new randomly chosen vector $\vec{v}$ modulo p. Then, with all but negligible probability, we have that choosing A_{k+1} modulo p so that $A_{k+1}A_kA_{k-1}A\vec{w}$ is the all zeros vector for a new randomly chosen vector $\vec{w}$ modulo p is equivalent (up to a negligible difference) to choosing $\vec{a}_{k+1}$ uniformly at random and choosing $\vec{b}_{k+1}$ randomly up to following constraints:*

1. *$\vec{b}_{k+1} \cdot \vec{a}_{k-2} = 0$ modulo p,*
2. *$\vec{b}_{k+1} \cdot \vec{a}_{k-1} = 0$ modulo p,*
3. *$\vec{b}_{k+1} \cdot \vec{a}_k = -1$ modulo p.*

Proof. With all but negligible probability, A is a rank $n-1$ matrix with left nullspace equal to the span of the vector $\left(a_1^{k-2}, \ldots, a_{n-1}^{k-2}, -1\right)$. Now, $A_{k-1}A$ is a rank $n-2$ matrix with a 2-dimensional left nullspace. We can alternatively think of the left nullspace as the kernel of $(A_{k-1}A)^T = A^T A_{k-1}^T$, where A^T denotes the transpose of A. It is clear that this kernel contains the kernel of A_{k-1}^T, which is equal to the span of the vector $\left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, -1\right)$. It also contains the vector $\left(a^{k-2}, \ldots, a_{n-1}^{k-2}, 0\right)$. To see this, note that $\vec{b}_{k-1} \cdot \vec{a}_k = -1$ modulo p, so

$$A_{k-1}^T \cdot \left(a^{k-2}, \ldots, a_{n-1}^{k-2}, 0\right)^T = \left(a_1^{k-2}, \ldots, a_{n-1}^{k-2}, -1\right),$$

which is in the kernel of A^T. With all but negligible probability, these vectors $\left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, -1\right)$ and $\left(a^{k-2}, \ldots, a_{n-1}^{k-2}, 0\right)$ are linearly independent and form a basis for the kernel of $(A_{k-1}A)^T$.

Now, by applying Lemma 13 to $C := A_{k-1}A$ and $\vec{b}_k$, we know that $\vec{b}_k$ is distributed randomly up to the constraints that $\vec{b}_k \cdot \vec{a}_{k-1} = -1$ modulo p and $\vec{b}_k \cdot \vec{a}_{k-2} = 0$ modulo p.

We now consider the kernel of $(A_k A_{k-1} A)^T = A^T A_{k-1}^T A_k^T$, which is a 3-dimensional space. It contains the kernel of A_k^T, which is equal to the span of $\left(a_1^k, \ldots, a_{n-1}^k, -1\right)$. We next observe that the vector $\left(a_1^{k-2}, \ldots, a_{n-1}^{k-2}, 0\right)$ is in the kernel of $A^T A_{k-1}^T A_k^T$. This holds because:

$$A_k^T \cdot \left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, 0\right)^T = \left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, 0\right),$$

which belongs to the kernel of $A^T A_{k-1}^T$. (Recall that $\vec{a}_{k-1} \cdot \vec{b}_k = 0$ modulo p.) We also observe that the vector $\left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, 0\right)$ is in the kernel of $A^T A_{k-1}^T A_k^T$. This holds because:

$$A_k^T \cdot \left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, 0\right)^T = \left(a_1^{k-1}, \ldots, a_{n-1}^{k-1}, -1\right),$$

since $\vec{b}_k \cdot \vec{a}_{k-1} = -1$ modulo p, and this is in the kernel of $A^T A_{k-1}^T$.

We now apply Lemma 13 again, this time with $C := A_k A_{k-1} A$. We conclude that choosing A_{k+1} so that the kernel of $A_{k+1} A_k A_{k-1} A$ includes a new random vector is equivalent (up to negligible difference) to choosing $\vec{a}_{k+1}$ uniformly at random and choosing $\vec{b}_{k+1}$ up to the constraints $\vec{b}_{k+1} \cdot \vec{a}_k = -1$, $\vec{b}_{k+1} \cdot \vec{a}_{k-1} = 0$, and $\vec{b}_{k+1} \cdot \vec{a}_{k-2} = 0$ modulo p. □

To prove that GameAlt$''_i$ and GameAlt$'_i$ are indistinguishable, we will use a hybrid argument over the four primes dividing N, applying Corollary 12 and Corollary 14 for each prime. To do this, we must define three additional games:

GameAlt$'_{i,1}$ This game is like GameAlt$'_i$, except that the $i-2$ update matrix is chosen so that there is a new random vector in the kernel of the matrix product modulo p_1. Essentially, this means that the $i-2$ update matrix is distributed as in GameAlt$''_i$ modulo p_1 and is distributed as in GameAlt$'_i$ modulo the other primes.

GameAlt$'_{i,2}$ This game is like GameAlt$'_{i,1}$, except that the $i-2$ update matrix is now chosen so that there is a new random vector in the kernel of the matrix product modulo p_2 as well. This means that the $i-2$ update matrix is distributed as in GameAlt$''_i$ modulo p_1, p_2 and is distributed as in GameAlt$'_i$ modulo p_3, p_4.

GameAlt$'_{i,3}$ This game is like GameAlt$'_{i,2}$, except that the $i-2$ update matrix is now chosen so that there is a new random vector in the kernel of the matrix product modulo p_3 as well. This means that the $i-2$ update matrix is distributed as in GameAlt$''_i$ modulo p_1, p_2, p_3 and is distributed as in GameAlt$'_i$ modulo p_4.

For convenience of notation, we can also let GameAlt$'_{i,0}$ be another name for GameAlt$'_i$ and let GameAlt$'_{i,4}$ be another name for GameAlt$''_i$. We then prove that GameAlt$'_i$ and GameAlt$''_i$ are indistinguishable by proving the following lemma:

Lemma 16. *For $\ell \leq (n-8)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between GameAlt$'_{i,j}$ and GameAlt$'_{i,j+1}$ with non-negligible advantage, for each i from 3 to $q+2$ and each j from 0 to 3.*

Proof. For $i = 3$, this statement holds trivially because GameAlt$''_3$ and GameAlt$'_3$ are exactly the same. We thus assume $i \geq 4$. We suppose there exists a PPT attacker $\mathcal{A}$ which can distinguish between GameAlt$'_{i,j}$ and GameAlt$'_{i,j+1}$ with non-negligible advantage. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $(\vec{\delta}, F(\vec{\tau}))$ and $(\vec{\delta}, F(\vec{\tau'}))$ from Corollary 12 with non-negligible probability. This will be a contradiction, since ϵ will be negligible.

$\mathcal{B}$ first chooses a bilinear group G of order $N = p_1p_2p_3p_4$, creates VK as specified by the KeyGen algorithm, and creates SK_0 as specified except that the G_{p_2} parts are distributed according to D_{Alt}. More precisely, the key is set as:

$$\vec{S_0} = g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}},\ \vec{U_0} = u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}},\ \vec{H_0} = h^{\vec{r}} g_2^{f_1\vec{c}+f_2\vec{d}} g_3^{\vec{z}},$$

where g, u, h are random elements of G_{p_1}, g_2 is a random element of G_{p_2}, g_3 is a random element of G_{p_3}, $\vec{r}$ is a random vector in $\mathbb{Z}_{p_1}^n$, $\vec{c}, \vec{d}$ are random vectors in $\mathbb{Z}_{p_2}^n$, f_1, f_2 are random values in $\mathbb{Z}_{p_2}$, and $\vec{x}, \vec{y}, \vec{z}$ are random vectors in $\mathbb{Z}_{p_3}$. We note that the factors p_1, p_2, p_3, p_4 are known to $\mathcal{B}$, as are all of the exponents $(\vec{r}, f_1, f_2, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z})$.

$\mathcal{B}$ gives the verification key VK to $\mathcal{A}$. For the first $i-4$ signature requests made by $\mathcal{A}$, $\mathcal{B}$ responds by running the signing algorithm and choosing the update matrix according to the prescribed distribution. We let A denote the product $A_{i-4} \cdots A_1$ (if $i = 4$, then A is the identity matrix).

Now, $\mathcal{B}$ receives the $i-3$ signature request from $\mathcal{A}$, along with its associated leakage function f_{i-3}. The current secret key is SK_{i-4}. It chooses the values $b_1^{i-3}, \ldots, b_{n-1}^{i-3}$ for A_{i-3} uniformly at random and chooses the values $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ uniformly at random modulo p_k for each $k \neq j+1$. (At this point, the only remaining variables are the values of $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ modulo p_{j+1}.) We let $\tilde{f}_{i-3}$ denote the function of the values $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ modulo p_{j+1} for A_{i-3} obtained by considering $f_{i-3}(A_{i-3}, \mathrm{SK}_{i-4})$ for these fixed values of $b_1^{i-3}, \ldots, b_{n-1}^{i-3}$ modulo N, $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ modulo p_k's for $k \neq j+1$, and SK_{i-4}.

$\mathcal{B}$ then receives a sample $\left(\vec{\delta}, F(\vec{\gamma})\right)$ as in the corollary, where $p = p_{j+1}$, $m := n-1$, and F is defined as follows. First, $\mathcal{B}$ chooses $\vec{t_1}, \vec{t_2}, \vec{t_3}$ to be three nonzero vectors which include the nonzero exponent vectors of the current secret key modulo p in the G_p subgroup (For example, if $p = p_3$, the exponent vectors are $A\vec{x}$, $A\vec{y}$, and $A\vec{z}$. If $p = p_4$, the exponent vectors are all zeros, so $\vec{t_1}$, $\vec{t_2}$, $\vec{t_3}$ are chosen to be arbitrary nonzero vectors.) $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^5$ is defined by:

$$F(\vec{\gamma}) := \left(\tilde{f}_{i-3}(\vec{\gamma}), \vec{\gamma} \cdot \vec{t_1}, \vec{\gamma} \cdot \vec{t_2}, \vec{\gamma} \cdot \vec{t_3}, \vec{\gamma} \cdot (A\vec{c}), \vec{\gamma} \cdot (A\vec{d}\,)\right).$$

(Note that when $p = p_3$, the three vectors $\vec{t_1}, \vec{t_2}, \vec{t_3}, A\vec{c}, A\vec{d}$ are all linearly independent, so it is necessary to give out all of these dot products with $\vec{\gamma}$ to enable $\mathcal{B}$ to compute SK_{i-3} and complete the subsequent steps. For the other primes, linear dependencies would allow us to give out fewer dot products, but we ignore this simplification.) In the notation of the corollary, this means that $|W| = 2^\ell \cdot p^5$. $\mathcal{B}$'s task is to distinguish whether $\vec{\gamma}$ satisfies $\vec{\gamma} \cdot \vec{\delta} = 0$ modulo p or not.

$\mathcal{B}$ will implicitly set the values $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ modulo p_{j+1} of the matrix A_{i-3} to be equal to $\gamma_1, \ldots, \gamma_{n-1}$. It provides $\mathcal{A}$ with $f_{i-3}(A_{i-3}, \mathrm{SK}_{i-4}) = \tilde{f}_{i-3}(\vec{\gamma})$. It can compute SK_{i-3} (and hence also the requested signature) by using its knowledge of SK_{i-4}, the values $b_1^{i-3}, \ldots, b_{n-1}^{i-3}$ for A_{i-3}, the value $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ modulo p_k for $k \neq j+1$, and the values $\vec{\gamma} \cdot \vec{t_1}, \vec{\gamma} \cdot \vec{t_2}, \vec{\gamma} \cdot \vec{t_3}$ modulo p_{j+1}. It is important here to note that this is all the information about $\vec{\gamma}$ that is needed to compute SK_{i-3}, because of how the $a_1^{i-3}, \ldots, a_{n-1}^{i-3}$ values appear in the matrix A_{i-3}. For this reason, $\mathcal{B}$ will not fully know A_{i-3}, but it will know SK_{i-3} (allowing it to continue producing signatures for the rest of the simulation because it will fully know all of the subsequent update matrices).

Next, $\mathcal{B}$ receives the $i-2$ signature request from $\mathcal{A}$, along with its associated leakage function f_{i-2}. $\mathcal{B}$ will choose the update matrix A_{i-2} as follows. With all but negligible probability, $\vec{\gamma} \cdot \vec{t_1} \neq 0$ modulo p (recall that $\vec{t_1}$ was chosen to be nonzero), and since $\mathcal{B}$ knows $\vec{t_1}$ and this dot product, it can multiply $\vec{t_1}$ by a suitable constant modulo p to obtain a vector $\vec{b'} \in \mathbb{Z}_p^{n-1}$ such that $\vec{\gamma} \cdot \vec{b'} = -1$ modulo p. It will then set the entries $b_1^{i-2}, \ldots, b_{n-1}^{i-2}$ for A_{i-2} as $b_j^{i-2} = \delta_j + b'_j$ modulo p, where δ_j denotes the j^{th} entry of $\vec{\delta}$ and b'_j denotes the j^{th} entry of $\vec{b'}$. Now, if $\vec{\delta}, \vec{\gamma}$ are both uniformly random modulo p, this means that $\vec{b}_{i-2} := (b_1^{i-2}, \ldots, b_{n-1}^{i-2})$ is also uniformly random. If $\vec{\delta}, \vec{\gamma}$ are random up to the constraint that $\vec{\delta} \cdot \vec{\gamma} = 0$, then $\vec{b}_{i-2}$ is distributed as a random vector up to the constraint that $\vec{b}_{i-2} \cdot \vec{\gamma} = -1$ modulo p. By Corollary 14, this means that the $i-2$ update will be properly distributed as in GameAlt$''_i$ modulo $p = p_{j+1}$ when $\vec{\delta} \cdot \vec{\gamma} = 0$, and will be properly distributed as in GameAlt$'_i$ modulo p when $\vec{\delta}, \vec{\gamma}$ are uniformly random. For p_k's where $k < j+1$, $\mathcal{B}$ will set the values of $b_1^{i-2}, \ldots, b_{n-1}^{i-2}$ modulo p_k to satisfy $\vec{b}_{i-2} \cdot \vec{a}_{i-3} = -1$ modulo p_k, where $\vec{a}_{i-3}$ denotes the entries in the final row of A_{i-3} modulo p_k. For $k > j+1$, it will choose the values of $b_1^{i-2}, \ldots, b_{n-1}^{i-2}$ randomly modulo p_k. It chooses the entries for the final row of A_{i-2} randomly.

For the $i-1$ signature request, $\mathcal{B}$ will choose a random vector $\vec{w}$ from the span of $\vec{c}, \vec{d}$. Since $\mathcal{B}$ knows the values $\vec{\gamma} \cdot (A\vec{c})$ and $\vec{\gamma} \cdot (A\vec{d}\,)$ modulo p, it can compute $A_{i-3}A\vec{w}$ modulo p. Since it knows A_{i-2}, it can then compute $A_{i-2}A_{i-3}A\vec{w}$ modulo p. (It can also compute this modulo the other primes, since it knows all the entries of A_{i-3} modulo the primes not equal to p_{j+1}.) It chooses the values $a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ modulo N for A_{i-1} randomly, and chooses the values $b_1^{i-1}, \ldots, b_{n-1}^{i-1}$ modulo N so that $A_{i-2}A_{i-3}A\vec{w}$ is in the kernel of A_{i-1} modulo N.

We let $\vec{v}$ denote a random vector such that the span of $\vec{v}, \vec{w}$ is equal to the span of $\vec{c}, \vec{d}$. $\mathcal{B}$ chooses the i^{th} update matrix A_i randomly up to the constraint that $A_{i-1}A_{i-2}A_{i-3}A\vec{v}$ is in the kernel of A_i modulo N. This cancels out all of the G_{p_2} parts from the secret key.

For the remaining updates, $\mathcal{B}$ chooses the update matrix according to the distribution specified in the update algorithm. If $\vec{\delta} \cdot \vec{\gamma} = 0$ modulo p, then $\mathcal{B}$ has properly simulated GameAlt$'_{i,j+1}$. If $\vec{\delta} \cdot \vec{\gamma} \neq 0$, then $\mathcal{B}$ has properly simulated GameAlt$'_{i,j}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish these two distributions with non-negligible probability. This will contradict Corollary 12 as long as ϵ is negligible.

To apply the corollary, we need:

$$|W| = 2^{\ell} p^5 \leq 4(1 - 1/p)p^{n-3}\epsilon^2,$$

so it suffices to have ℓ such that

$$\ell \leq (n-8)\log(p_{j+1}) + 2\log(\epsilon)$$

for some negligible ϵ. For simplicity, we define $\delta = -\log(\epsilon)$. We then obtain the desired result as long as

$$\ell \leq (n-8)\log(p_{j+1}) - 2\delta,$$

for any δ such that $2^{-\delta}$ is negligible. □

As an immediate consequence of Lemma 16, we conclude:

Lemma 17. *When $\ell \leq (n-8)\log(p_j) - 2\delta$ for all p_j dividing N and for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between GameAlt$''_i$ and GameAlt$'_i$ with non-negligible advantage, for each i from 3 to $q+2$.*

We are now left with the task of showing that a PPT attacker $\mathcal{A}$ cannot distinguish between GameAlt$'_i$ and GameAlt$_i$. We prove:

Lemma 18. *When $\ell \leq (n-8)\log(p_j) - 2\delta$ for all p_j dividing N and for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between GameAlt$_i$ and GameAlt$'_i$ with non-negligible advantage, for each i from 3 to $q+2$.*

The proof of this lemma is bit long and intricate, but it essentially uses the same techniques as the proof of Lemma 17, combined with Lemma 11 and Corollary 15. The proof can be found in Appendix A.

5.6 Transitions from $\mathbf{GameAlt}_2$ to $\mathbf{GameAlt}_1$ and to $\mathbf{GameAlt}_0$

We also prove:

Lemma 19. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s probability of producing a Type II forgery between GameAlt$_2$ and GameAlt$_1$ is negligible, as long as $\ell \leq \frac{1}{2}(\log(p_2) - 2\delta)$, where $\delta > 0$ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Lemma 20. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s probability of producing a Type II forgery between GameAlt$_1$ and GameAlt$_0$ is negligible, as long as $\ell \leq \log(p_2) - 2\delta$, where $\delta > 0$ is a parameter chosen so that $2^{-\delta}$ is negligible.*

The proofs of these lemmas are similar to the proof of Lemma 9, and can be found in Appendices B and C.

Putting together the results of the previous subsections, we obtain Theorem 4.

6 Discussion

6.1 Our Leakage Parameter

As stated in our security theorems, the leakage ℓ we can allow is equal to the minimum of $\frac{1}{3}(\log(p_2) - 2\delta)$ and $(n-8)\log(p_j) - 2\delta$ for all p_j dividing N, where δ is chosen so that $2^{-\delta}$ is negligible. The bound $\frac{1}{3}(\log(p_2) - 2\delta)$ arises from our need to hide information about the secret key itself, while the bound $(n-8)\log(p_j) - 2\delta$ arises from our need to hide information about the update matrices. If we choose our primes $p_1, \ldots, p_4$ so that $\log(p_1), \log(p_3), \log(p_4)$ are approximately equal to some parameter κ, $\log(p_2)$ is approximately equal to 3κ, $n = 9$, and δ is much smaller than κ (say equal to $\epsilon\kappa$ for a small constant ϵ), then our parameter ℓ is approximately equal to κ. If the number of bits representing a group element of G is approximately $\log(N) = 6\kappa$, then our secret key has length $3n\log(N) = 162\kappa$ bits, and the variables in our update matrices can be represented with $2(n-1)\log(N) = 96\kappa$ bits. Thus, the amount of leakage allowed per update/signature is roughly a $\frac{1}{162}$ fraction of the secret key length, a $\frac{1}{96}$ fraction of the update randomness, and a $\frac{1}{258}$ fraction of the total length of the secret key and update randomness. This is a small constant, but it is still a positive constant independent of the security parameter, and we view this as an important qualitative (as well as a quantitative) step forward on the path to optimally leakage-resilient schemes. One might try to modestly improve this constant by finding a more optimized instantiation of our techniques.

6.2 Generalizing Our Approach

Our signature scheme can be viewed as a particular instantiation of a more general approach. In essence, the key generation algorithm samples some random bits R which are used to define a vector space $V_R^p \subseteq \mathbb{Z}_p^m$ for each prime p dividing N. A secret key which is compatible with the fixed public key corresponds to a collection of m group elements in G viewed as a base element raised to a vector, where the vector belongs to V_R^p for each prime p dividing N. The update matrices which are applied in the exponent are chosen from a class of linear transformations that map V_R^p into itself for each p. The proof of security generally proceeds by canceling one of these subspaces in the key at progressively earlier stages in the security game until it can be purged all together. We then must argue three things: 1) in the original game, the attacker cannot produce a forgery *without* this subspace, 2) in the final game, the attacker cannot produce a forgery *with* this subspace, and 3) the attacker cannot change its forgery type with non-negligible probability as we go through the sequence of games.

In the specific scheme we present, $m = 3n$, and the spaces V_R^p can be described as follows. We let e_i denote the vector in $\mathbb{Z}_p^m$ with a 1 in the i^{th} coordinate and 0's elsewhere. We consider the secret key as vector of $3n$ group elements where the elements of $\vec{S}_0$ are the first n entries, the elements of $\vec{U}_0$ are the second n entries, and the elements of $\vec{H}_0$ are the last n entries. Then, $V_R^{p_1}$ is the n dimensional subspace spanned by the vectors $e_i + \alpha e_{i+n} + \beta e_{i+2n}$ for $1 \leq i \leq n$. $V_R^{p_2}$ and $V_R^{p_3}$ are both $3n$ dimensional spaces (i.e. equal to $\mathbb{Z}_{p_2}^m$ and $\mathbb{Z}_{p_3}^m$ respectively), and $V_R^{p_4}$ contains only the all zeros vector.

One might try to execute our general approach with fewer prime factors for N. Previous results employing the dual system encryption methodology with three prime factors or in prime order groups ([36, 38, 46] for example) suggest that fewer primes would be sufficient. We chose to work with four prime factors for ease of exposition. In particular, one could imagine using a prime order group and specifying several subspaces V_R which are orthogonal to each other. (This is similar to the approach used by Freeman [25] for obtaining analogs in prime order bilinear groups of some previously proposed constructions in composite order bilinear groups.) However, we do not expect that natural analogs of our system obtained in such ways would produce a substantially improved constant for the leakage fraction.

6.3 Identity-Based Encryption and Other Advanced Functionalities

We might expect our techniques to extend naturally to the settings of Identity-Based Encryption, Hierarchical Identity-Based Encryption, Attribute-Based Encryption, and other advanced functionalities. One can view the work of [37] as heuristic evidence for this, since they provide leakage-resilient IBE/HIBE and ABE systems from dual system encryption techniques for the setting where updates are assumed to be leak-free.

6.4 Open Problems

There are two very interesting problems for signatures in the continual memory leakage model which remain open. First, while our scheme allows a constant fraction of the secret state to leak between and during updates, this fraction is extremely low (as noted above). Leaking a much better fraction (ideally $1 - o(1)$) is a worthy goal, and will most likely require significant new ideas. We note that a fraction of $1-o(1)$ for leakage *between* updates has been obtained in several previous works (e.g. [10, 12, 39] in the continual memory leakage model). Secondly, allowing any super-logarithmic amount of leakage during the initial key generation process remains an open problem.

7 Acknowledgement

We thank Yannis Rouselakis for helpful comments.

References

[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[2] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[3] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[4] M. Bellare, B. Waters, and S. Yilek. Identity-based encryption secure under selective opening attack. Cryptology ePrint Archive, Report 2010/159, 2010. `http://eprint.iacr.org/`.

[5] E. Biham, Y. Carmeli, and A. Shamir. Bug attacks. In *CRYPTO*, pages 221–240, 2008.

[6] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[7] D. Boneh and D. Brumley. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[8] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.

[9] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[10] E. Boyle, G. Segev, and D. Wichs. Fully leakage-resilient signatures. Cryptology ePrint Archive, Report 2010/488, 2010.

[11] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability (or: Quadratic residuosity strikes back). In *CRYPTO*, 2010.

[12] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Cryptography resilient to continual memory leakage. In *FOCS*, 2010.

[13] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453–469, 2000.

[14] D. Cash, Y. Z. Ding, Y. Dodis, W. Lee, R. J. Lipton, and S. Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In *TCC*, pages 479–498, 2007.

[15] S. Chow, Y. Dodis, Y. Rouselakis, and B. Waters. Practical leakage-resilient identity-based encryption from simple assumptions. In *ACM Conference on Computer and Communications Security*, 2010.

[16] D. Di Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, pages 225–244, 2006.

[17] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, 2010.

[18] Y. Dodis, Y. Kalai, and S. Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[19] Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT*, pages 301–324, 2001.

[20] S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, pages 207–224, 2006.

[21] S. Dziembowski and K. Pietrzak. Intrusion-resilient secret sharing. In *FOCS*, pages 227–237, 2007.

[22] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[23] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[24] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[25] D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT*, pages 44–61, 2010.

[26] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[27] M. Gerbush, A. Lewko, and B. Waters. Dual form signatures. Manuscript, 2010. Available at `http://www.cs.utexas.edu/~alewko/DualForm.pdf`.

[28] S. Goldwasser and G. Rothblum. How to play mental solitaire under continuous side-channels: A completeness theorem using secure hardware. In *CRYPTO*, 2010.

[29] A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Applebaum, and E. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[30] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[31] A. Juma and Y. Vahlis. On protecting cryptographic keys against side-channel attacks. In *CRYPTO*, 2010.

[32] J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In *FOCS*, pages 92–101, 2003.

[33] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[34] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[35] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[36] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[37] A. Lewko, Y. Rouselakis, and B. Waters. Achieving leakage resilience through dual system encryption. Cryptology ePrint Archive, Report 2010/438, 2010.

[38] A. Lewko and B. Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In *TCC*, pages 455–479, 2010.

[39] T. Malkin, I. Teranishiy, Y. Vahlis, and M. Yung. Signatures resilient to continual leakage on memory and computation. Cryptology ePrint Archive, Report 2010/522, 2010.

[40] S. Micali and L. Reyzin. Physically observable cryptography. In *TCC*, pages 278–296, 2004.

[41] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[42] P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.

[43] C. Petit, F.X. Standaert, O. Pereira, T. Malkin, and M. Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. In *ASIACCS*, pages 56–65, 2008.

[44] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[45] J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[46] B. Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

A Proof of Lemma 18

Again, we will use a hybrid argument over the primes dividing N, so we begin by defining the following additional games.

GameAlt$_{i,1}$ This game is like GameAlt$'_i$, except that the $i+1$ update matrix is chosen so that there is a new random vector in the kernel of the matrix product modulo p_1. Essentially, this means that the $i+1$ update matrix is distributed as in GameAlt$_i$ modulo p_1 and is distributed as in GameAlt$'_i$ modulo the other primes.

GameAlt$_{i,2}$ This game is like GameAlt$_{i,1}$, except that the $i+1$ update matrix is now chosen so that there is a new random vector in the kernel of the matrix product modulo p_2 as well. This means that the $i+2$ update matrix is distributed as in GameAlt$_i$ modulo p_1, p_2 and is distributed as in GameAlt$'_i$ modulo p_3, p_4.

GameAlt$_{i,3}$ This game is like GameAlt$_{i,2}$, except that the $i+1$ update matrix is now chosen so that there is a new random vector in the kernel of the matrix product modulo p_3 as well. This means that the $i+1$ update matrix is distributed as in GameAlt$_i$ modulo p_1, p_2, p_3 and is distributed as in GameAlt$'_i$ modulo p_4.

For convenience of notation, we can also let GameAlt$_{i,0}$ be another name for GameAlt$'_i$ and let GameAlt$_{i,4}$ be another name for GameAlt$_i$. We will prove that GameAlt$'_i$ and GameAlt$_i$ are indistinguishable by proving the following lemma:

Lemma 21. *For $\ell \leq (n-8)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between $GameAlt_{i,j}$ and $GameAlt_{i,j+1}$ with non-negligible advantage, for each i from 3 to $q+2$ and each j from 0 to 3.*

As a consequence of Corollary 15, we know that choosing the $i+1$ update matrix so that there is a new random vector in the kernel of the matrix product modulo a prime p is equivalent to choosing the vector $\vec{b}_{i+1}$ satisfying three dot product conditions modulo p. Namely,

1. $\vec{b}_{i+1} \cdot \vec{a}_{i-2} = 0$ modulo p,
2. $\vec{b}_{i+1} \cdot \vec{a}_{i-1} = 0$ modulo p
3. $\vec{b}_{i+1} \cdot \vec{a}_{i} = -1$ modulo p.

We could prove Lemma 21 directly by an application of Lemma 11 with $d = 3$, but setting such a high value of d would make our leakage parameter considerably worse (note the dependence of the exponent on d in the lemma). Instead, we will obtain a better leakage parameter by proving this lemma in a few steps. Essentially, we will employ a hybrid argument over the three conditions enumerated above. We are able to do this because the attacker can only obtain leakage on each of $\vec{a}_{i-2}$, $\vec{a}_{i-1}$, and $\vec{a}_i$ separately and in the proper order. We introduce two additional games between each $\text{GameAlt}_{i,j}$ and $\text{GameAlt}_{i,j+1}$:

GameAlt$_{i,j,1}$ This game is like $\text{GameAlt}_{i,j}$, except that the $i+1$ update matrix is chosen modulo p_{j+1} so that its vector $\vec{b}_{i+1}$ is a random vector satisfying condition 1. above.

GameAlt$_{i,j,2}$ This game is like $\text{GameAlt}_{i,j,1}$, except that the $i+1$ update matrix is chosen modulo p_{j+1} so that its vector $\vec{b}_{i+1}$ is now a random vector satisfying conditions 1. and 2. above.

We now prove:

Lemma 22. *For $\ell \leq (n-8)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between $GameAlt_{i,j}$ and $GameAlt_{i,j,1}$ with non-negligible advantage, for each i from 3 to $q+2$ and each j from 0 to 3.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which can distinguish between $\text{GameAlt}_{i,j}$ and $\text{GameAlt}_{i,j,1}$ with non-negligible advantage. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $(\vec{\delta}, F(\vec{\tau}))$ and $(\vec{\delta}, F(\vec{\tau}'))$ from Corollary 12 with non-negligible probability. This will contradict the corollary, since ϵ will be negligible.

$\mathcal{B}$ first chooses a bilinear group G of order $N = p_1p_2p_3p_4$, creates VK as specified by the KeyGen algorithm, and creates SK_0 as specified except that the G_{p_2} parts are distributed according to D_{Alt}. More precisely, the key is set as:

$$\vec{S}_0 = g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}},\ \vec{U}_0 = u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}},\ \vec{H}_0 = h^{\vec{r}} g_2^{f_1\vec{c}+f_2\vec{d}} g_3^{\vec{z}},$$

where g, u, h are random elements of G_{p_1}, g_2 is a random element of G_{p_2}, g_3 is a random element of G_{p_3}, $\vec{r}$ is a random vector in $\mathbb{Z}_{p_1}^n$, $\vec{c}, \vec{d}$ are random vectors in $\mathbb{Z}_{p_2}^n$, f_1, f_2 are random values in $\mathbb{Z}_{p_2}$, and $\vec{x}, \vec{y}, \vec{z}$ are random vectors in $\mathbb{Z}_{p_3}$. We note that the factors p_1, p_2, p_3, p_4 are known to $\mathcal{B}$, as are all of the exponents $(\vec{r}, f_1, f_2, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z})$.

$\mathcal{B}$ gives the verification key VK to $\mathcal{A}$. For the first $i-3$ signature requests made by $\mathcal{A}$, $\mathcal{B}$ responds by running the signing algorithm and choosing the update matrix according to the prescribed distribution. We let A denote the product $A_{i-3} \cdots A_1$ (if $i = 3$, then A is the identity matrix). The current secret key is SK_{i-3}.

Next, $\mathcal{B}$ receives the $i-2$ signature request from $\mathcal{A}$, along with the associated leakage function f_{i-2} to be applied to A_{i-2} and SK_{i-3}. $\mathcal{B}$ chooses the values $b_1^{i-2}, \ldots, b_{n-1}^{i-2}$ randomly modulo N, and chooses the values $a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ randomly modulo the prime factors of N not equal to p_{j+1}. The only remaining variables in A_{i-2} are the values of $a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ modulo p_{j+1}. We let $\tilde{f}_{i-2}$ be the function of these variables obtained by considering $f_{i-2}(\mathrm{SK}_{i-3}, A_{i-2})$ with all the other values fixed.

$\mathcal{B}$ then receives a sample $\left(\vec{\delta}, F(\vec{\gamma})\right)$ as in Corollary 12, where $p = p_{j+1}$, $m := n-1$, and F is defined as follows. We let $\vec{t}_1, \vec{t}_2, \vec{t}_3$ denote the exponent vectors of the current secret key modulo p in the G_p subgroup (For example, if $p = p_3$, these are $A\vec{x}$, $A\vec{y}$, and $A\vec{z}$.) $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^5$ is defined by:

$$F(\vec{\gamma}) := \left(\tilde{f}_{i-2}(\vec{\gamma}), \vec{\gamma}\cdot\vec{t}_1, \vec{\gamma}\cdot\vec{t}_2, \vec{\gamma}\cdot\vec{t}_3, \vec{\gamma}\cdot(A\cdot\vec{c}), \vec{\gamma}\cdot(A\cdot\vec{d}\,)\right).$$

(Again, some of these may be redundant for particular values of $j+1$, but we ignore this.) In the notation of the corollary, this means that $|W| = 2^\ell \cdot p^5$. $\mathcal{B}$'s task is to distinguish whether $\vec{\gamma}$ is a random vector from the orthogonal space to $\vec{\delta}$ or a uniformly random vector modulo p.

$\mathcal{B}$ will implicitly set the values $a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ modulo p_{j+1} of the matrix A_{i-2} to be equal to $\gamma_1, \ldots, \gamma_{n-1}$. It then provides $\mathcal{A}$ with $f_{i-2}(A_{i-2}, \mathrm{SK}_{i-3}) = \tilde{f}_{i-2}(\vec{\gamma})$. It can compute SK_{i-2} by using its knowledge of SK_{i-3}, the values $b_1^{i-2}, \ldots, b_{n-1}^{i-2}$ for A_{i-2}, the values $a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ modulo p_k for $k \neq j+1$, and the values $\vec{\gamma}\cdot\vec{t}_1, \vec{\gamma}\cdot\vec{t}_2, \vec{\gamma}\cdot\vec{t}_3$ modulo p_{j+1}. It is important here to note that this is all the information about $\vec{\gamma}$ that is needed to compute SK_{i-2}, because of how the $a_1^{i-2}, \ldots, a_{n-1}^{i-2}$ values appear in the matrix A_{i-2}. We note that $\mathcal{B}$ will not fully know A_{i-2}, but it will know SK_{i-2} (allowing it to produce the requested signature here and to continue producing signatures for the rest of the simulation because it will fully know all of the subsequent update matrices).

Next, $\mathcal{B}$ receives the $i-1$ signature request from $\mathcal{A}$, along with the associated leakage function f_{i-1}. $\mathcal{B}$ chooses the values $a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ uniformly at random modulo N. $\mathcal{B}$ then chooses a random vector $\vec{w}$ from the span of $\vec{c}, \vec{d}$. Since $\mathcal{B}$ knows the values $\vec{\gamma}\cdot(A\vec{c})$ and $\vec{\gamma}\cdot(A\vec{d}\,)$ modulo p, it can compute $A_{i-2}A\vec{w}$ modulo p. (It can also compute this modulo the other primes, since it knows all the entries of A_{i-2} modulo the primes not equal to p_{j+1}.) It chooses the values $b_1^{i-1}, \ldots, b_{n-1}^{i-1}$ modulo N so that $A_{i-2}A\vec{w}$ is in the kernel of A_{i-1} modulo N. In other words, it sets:

$$b_j^{i-1} = -\frac{w'_j}{w'_n},$$

where $\vec{w}'$ denotes the vector $A_{i-2}A\vec{w}$. (Note that w'_n is invertible modulo N with all but negligible probability.) Since $\mathcal{B}$ knows the matrix A_{i-1} and the secret key SK_{i-2}, it can easily provide $\mathcal{A}$ with the leakage $f_{i-1}(A_{i-1}, \mathrm{SK}_{i-2})$ as well as the requested signature.

We let $\vec{v}$ denote a random vector such that the span of $\vec{v}, \vec{w}$ is equal to the span of $\vec{c}, \vec{d}$. $\mathcal{B}$ chooses the i^{th} update matrix A_i so that $A_{i-1}A_{i-2}A\vec{v}$ is in the kernel of A_i modulo N. This cancels out all of the G_{p_2} parts from the secret key. $\mathcal{B}$ can compute $A_{i-1}A_{i-2}A\vec{v}$ modulo N because it knows the values $\vec{\gamma}\cdot(A\vec{c})$ and $\vec{\gamma}\cdot(A\vec{d}\,)$ modulo p as well as the matrix A_{i-1}. Now, A_i is fully known to $\mathcal{B}$, so it can easily answer the leakage query here.

To choose the update matrix A_{i+1}, $\mathcal{B}$ chooses $a_1^{i+1}, \ldots, a_{n-1}^{i+1}$ uniformly at random modulo N, and chooses the values $b_1^{i+1}, \ldots, b_{n-1}^{i+1}$ modulo prime factors of N not equal to p_{j+1} such that $\vec{b}_{i+1}$ satisfies the dot product conditions 1., 2., and 3. modulo primes p_k for $k \leq j$, and is random for primes p_k for $k > j+1$. Modulo p_{j+1}, $\mathcal{B}$ sets $\vec{b}_{i+1}$ equal to $\vec{\delta}$.

For the remaining updates, $\mathcal{B}$ chooses the update matrices according to the distribution specified in the update algorithm. If $\vec{\gamma}\cdot\vec{\delta} = 0$ modulo p, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j,1}$. If $\vec{\gamma}$ and $\vec{\delta}$ are independently random, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j}$.

Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish these two distributions with non-negligible probability. This will contradict Corollary 12 as long as ϵ is negligible.

To apply the corollary, we need:

$$|W| = 2^{\ell} p^5 \leq 4(1 - 1/p)p^{n-3}\epsilon^2,$$

so it suffices to have ℓ such that

$$\ell \leq (n - 8)\log(p_{j+1}) + 2\log(\epsilon)$$

for some negligible ϵ. For simplicity, we define $\delta = -\log(\epsilon)$. We then obtain the desired result as long as

$$\ell \leq (n - 8)\log(p_{j+1}) - 2\delta,$$

for any δ such that $2^{-\delta}$ is negligible.

□

Lemma 23. *For $\ell \leq (n-8)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between GameAlt$_{i,j,1}$ and GameAlt$_{i,j,2}$ with non-negligible advantage, for each i from 3 to $q+2$ and each j from 0 to 3.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which can distinguish between GameAlt$_{i,j,1}$ and GameAlt$_{i,j,2}$ with non-negligible advantage. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $(X, F(X \cdot T))$ and $(X, F(Y))$ from Lemma 11 with non-negligible probability. We will set the parameters of the lemma as: $m := n-1$, $k := n-3$, and $d := 1$. This will contradict the lemma, since ϵ will be negligible.

$\mathcal{B}$ first chooses a bilinear group G of order $N = p_1p_2p_3p_4$, creates VK as specified by the KeyGen algorithm, and creates SK$_0$ as specified except that the G_{p_2} parts are distributed according to D_{Alt}. More precisely, the key is set as:

$$\vec{S_0} = g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}},\ \vec{U_0} = u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}},\ \vec{H_0} = h^{\vec{r}} g_2^{f_1\vec{c}+f_2\vec{d}} g_3^{\vec{z}},$$

where g, u, h are random elements of G_{p_1}, g_2 is a random element of G_{p_2}, g_3 is a random element of G_{p_3}, $\vec{r}$ is a random vector in $\mathbb{Z}_{p_1}^n$, $\vec{c}, \vec{d}$ are random vectors in $\mathbb{Z}_{p_2}^n$, f_1, f_2 are random values in $\mathbb{Z}_{p_2}$, and $\vec{x}, \vec{y}, \vec{z}$ are random vectors in $\mathbb{Z}_{p_3}$. We note that the factors p_1, p_2, p_3, p_4 are known to $\mathcal{B}$, as are all of the exponents $(\vec{r}, f_1, f_2, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z})$.

$\mathcal{B}$ gives the verification key VK to $\mathcal{A}$. For the first $i-2$ signature requests made by $\mathcal{A}$, $\mathcal{B}$ responds by running the signing algorithm and choosing the update matrix according to the prescribed distribution. We let A denote the product $A_{i-2} \cdots A_1$. The current secret key is SK$_{i-2}$.

Next, $\mathcal{B}$ receives the $i-1$ signature request from $\mathcal{A}$, along with the associated leakage function f_{i-1} to be applied to A_{i-1} and SK$_{i-2}$. $\mathcal{B}$ chooses a random vector $\vec{w}$ from the span of $\vec{c}, \vec{d}$. It then chooses the values $b_1^{i-1}, \ldots, b_{n-1}^{i-1}$ so that $A\vec{w}$ is in the kernel of A_{i-1}. We let $\vec{v}$ denote a random vector such that the span of $\vec{v}$ and $\vec{w}$ is equal to the span of $\vec{c}$ and $\vec{d}$.

$\mathcal{B}$ then chooses the values $a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ randomly modulo the prime factors of N not equal to p_{j+1}. The only remaining variables in A_{i-1} are the values of $a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ modulo p_{j+1}. We let $\tilde{f}_{i-1}$ be the function of these variables obtained by considering $f_{i-1}(\text{SK}_{i-2}, A_{i-1})$ with all the other values fixed.

$\mathcal{B}$ then receives a sample $(X, F(\vec{\gamma}))$ as in Lemma 11, where $p = p_{j+1}$, $m := n-1$, $k := n-3$, $d = 1$, and F is defined as follows. We let $\vec{t_1}, \vec{t_2}, \vec{t_3}$ denote the exponent vectors of the current secret key modulo p in the G_p subgroup (For example, if $p = p_3$, these are $A\vec{x}$, $A\vec{y}$, and $A\vec{z}$.) $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^4$ is defined by:

$$F(\vec{\gamma}) := \left(\tilde{f}_{i-1}(\vec{\gamma}), \vec{\gamma} \cdot \vec{t_1}, \vec{\gamma} \cdot \vec{t_2}, \vec{\gamma} \cdot \vec{t_3}, \vec{\gamma} \cdot (A\vec{v})\right).$$

In the notation of the lemma, this means that $|W| = 2^\ell \cdot p^4$. $\mathcal{B}$'s task is to distinguish whether $\vec{\gamma}$ is a random vector from the column space of X or a uniformly random vector modulo p.

$\mathcal{B}$ will implicitly set the values $a_1^{i-1}, \dots, a_{n-1}^{i-1}$ modulo p_{j+1} of the matrix A_{i-1} to be equal to $\gamma_1, \dots, \gamma_{n-1}$. It then provides $\mathcal{A}$ with $f_{i-1}(A_{i-1}, \mathrm{SK}_{i-2}) = \tilde{f}_{i-1}(\vec{\gamma})$. It can compute SK_{i-1} by using its knowledge of SK_{i-2}, the values $b_1^{i-1}, \dots, b_{n-1}^{i-1}$ for A_{i-1}, the values $a_1^{i-1}, \dots, a_{n-1}^{i-1}$ modulo p_k for $k \neq j+1$, and the values $\vec{\gamma} \cdot \vec{t_1}, \vec{\gamma} \cdot \vec{t_2}, \vec{\gamma} \cdot \vec{t_3}$ modulo p_{j+1}. We note that $\mathcal{B}$ will not fully know A_{i-1}, but it will know SK_{i-1} (allowing it to produce the requested signature here and to continue producing signatures for the rest of the simulation because it will fully know all of the subsequent update matrices).

Next, $\mathcal{B}$ receives the i signature request from $\mathcal{A}$, along with the associated leakage function f_i. $\mathcal{B}$ chooses the values $a_1^i, \dots, a_{n-1}^i$ uniformly at random and chooses the values $b_1^i, \dots, b_{n-1}^i$ so that $\vec{v}$ is in the kernel of $A_i A_{i-1} A$. Since $\mathcal{B}$ knows the matrix A_i and the secret key SK_{i-1}, it can easily provide $\mathcal{A}$ with the leakage $f_i(A_i, \mathrm{SK}_{i-1})$ as well as the requested signature.

To choose the update matrix A_{i+1}, $\mathcal{B}$ chooses $a_1^{i+1}, \dots, a_{n-1}^{i+1}$ uniformly at random modulo N, and chooses the values $b_1^{i+1}, \dots, b_{n-1}^{i+1}$ modulo prime factors of N not equal to p_{j+1} such that $\vec{b}_{i+1}$ satisfies the dot product conditions 1., 2., and 3. modulo primes p_k for $k \leq j$, and is random for primes p_k for $k > j+1$. Modulo p_{j+1}, $\mathcal{B}$ sets $\vec{b}_{i+1}$ as follows. We note that the space of vectors $\in \mathbb{Z}_p^{n-1}$ which are orthogonal to the column space of X is a (uniformly random) space of dimension 2. $\mathcal{B}$ samples the vector $\vec{b}_{i+1}$ modulo p randomly from the intersection of this with the $n-2$ dimensional space of vectors which are orthogonal to $\vec{a}_{i-2}$ (this intersection is non-trivial because we are working in the $n-1$ dimensional space $\mathbb{Z}_p^{n-1}$). This ensures that $\vec{b}_{i+1}$ satisfies condition 1. modulo p.

For the remaining updates, $\mathcal{B}$ chooses the update matrix according to the distribution specified in the update algorithm. If $\vec{\gamma}$ is a uniformly random vector modulo p, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j,1}$. To see this, note that $\vec{b}_{i+1}$ is distributed as a random vector up to satisfying condition 1. modulo p, since the space X is chosen randomly.

If $\vec{\gamma}$ is a random vector from the column space of X, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j,2}$. To see this, note that $\vec{b}_{i+1}$ is orthogonal to both $\vec{a}_{i-2}$ and $\vec{a}_{i-1} = \vec{\gamma}$ modulo p, and so satisfies conditions 1. and 2.. We further have that $\vec{b}_{i+1}$ is distributed randomly up to these conditions, because X is distributed randomly up to the constraints that it is orthogonal to $\vec{b}_{i+1}$ and contains $\vec{\gamma}$.

Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish these two distributions with non-negligible probability. This will contradict Lemma 11 as long as ϵ is negligible.

To apply the lemma, we need:

$$|W| = 2^\ell p^4 \leq 4(1 - 1/p)p^{n-4}\epsilon^2,$$

so it suffices to have ℓ such that

$$\ell \leq (n-8)\log(p_{j+1}) + 2\log(\epsilon)$$

for some negligible ϵ. For simplicity, we define $\delta = -\log(\epsilon)$. We then obtain the desired result as long as

$$\ell \leq (n-8)\log(p_{j+1}) - 2\delta,$$

for any δ such that $2^{-\delta}$ is negligible. □

Lemma 24. *For $\ell \leq (n-8)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker $\mathcal{A}$ can distinguish between GameAlt$_{i,j,2}$ and GameAlt$_{i,j+1}$ with non-negligible advantage, for each i from 3 to $q+2$ and each j from 0 to 3.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which can distinguish between GameAlt$_{i,j,2}$ and GameAlt$_{i,j+1}$ with non-negligible advantage. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $(X, F(X \cdot T))$ and $(X, F(Y))$ from Lemma 11 with non-negligible probability. We will set the parameters of the lemma as: $m := n-1$, $k := n-4$, and $d := 1$. This will contradict the lemma, since ϵ will be negligible.

$\mathcal{B}$ first chooses a bilinear group G of order $N = p_1 p_2 p_3 p_4$, creates VK as specified by the KeyGen algorithm, and creates SK$_0$ as specified except that the G_{p_2} parts are distributed according to D_{Alt}. More precisely, the key is set as:

$$\vec{S_0} = g^{\vec{r}} g_2^{\vec{c}} g_3^{\vec{x}},\ \vec{U_0} = u^{\vec{r}} g_2^{\vec{d}} g_3^{\vec{y}},\ \vec{H_0} = h^{\vec{r}} g_2^{f_1 \vec{c} + f_2 \vec{d}} g_3^{\vec{z}},$$

where g, u, h are random elements of G_{p_1}, g_2 is a random element of G_{p_2}, g_3 is a random element of G_{p_3}, $\vec{r}$ is a random vector in $\mathbb{Z}_{p_1}^n$, $\vec{c}, \vec{d}$ are random vectors in $\mathbb{Z}_{p_2}^n$, f_1, f_2 are random values in $\mathbb{Z}_{p_2}$, and $\vec{x}, \vec{y}, \vec{z}$ are random vectors in $\mathbb{Z}_{p_3}$. We note that the factors p_1, p_2, p_3, p_4 are known to $\mathcal{B}$, as are all of the exponents $(\vec{r}, f_1, f_2, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z})$.

$\mathcal{B}$ gives the verification key VK to $\mathcal{A}$. For the first $i-2$ signature requests made by $\mathcal{A}$, $\mathcal{B}$ responds by running the signing algorithm and choosing the update matrix according to the prescribed distribution. We let A denote the product $A_{i-2} \cdots A_1$. The current secret key is SK$_{i-2}$.

Next, $\mathcal{B}$ receives the $i-1$ signature request from $\mathcal{A}$, along with the associated leakage function f_{i-1} to be applied to A_{i-1} and SK$_{i-2}$. $\mathcal{B}$ chooses a random vector $\vec{w}$ from the span of $\vec{c}, \vec{d}$. It then chooses the values $b_1^{i-1}, \ldots, b_{n-1}^{i-1}$ so that $A\vec{w}$ is in the kernel of A_{i-1}. We let $\vec{v}$ denote a random vector such that the span of $\vec{v}$ and $\vec{w}$ is equal to the span of $\vec{c}$ and $\vec{d}$. $\mathcal{B}$ chooses the values $a_1^{i-1}, \ldots, a_{n-1}^{i-1}$ uniformly at random modulo N. It knows SK$_{i-2}$ and A_{i-1}, so it can easily compute $f_{i-1}(\text{SK}_{i-2}, A_{i-1})$ as well as the requested signature and gives these to $\mathcal{A}$. Next, $\mathcal{B}$ receives the i signature request from $\mathcal{A}$, along with the associated leakage function f_i. $\mathcal{B}$ chooses the values $b_1^i, \ldots, b_{n-1}^i$ so that $\vec{v}$ is in the kernel of $A_i A_{i-1} A$.

$\mathcal{B}$ then chooses the values $a_1^i, \ldots, a_{n-1}^i$ randomly modulo the prime factors of N not equal to p_{j+1}. The only remaining variables in A_i are the values of $a_1^i, \ldots, a_{n-1}^i$ modulo p_{j+1}. We let $\tilde{f}_i$ be the function of these variables obtained by considering $f_i(\text{SK}_{i-1}, A_i)$ with all the other values fixed.

$\mathcal{B}$ then receives a sample $(X, F(\vec{\gamma}))$ as in Lemma 11, where $p = p_{j+1}$, $m := n-1$, $k := n-4$, $d = 1$, and F is defined as follows. $\mathcal{B}$ chooses $\vec{t}_1, \vec{t}_2, \vec{t}_3$ to be three linearly independent vectors whose span includes the exponent vectors of the current secret key modulo p in the G_p subgroup (we note the coefficients needed to express the exponent vectors in terms of $\vec{t}_1, \vec{t}_2, \vec{t}_3$ will be known to $\mathcal{B}$). (For example, if $p = p_3$, the span includes $A\vec{x}$, $A\vec{y}$, and $A\vec{z}$.) $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^3$ is defined by:

$$F(\vec{\gamma}) := \left(\tilde{f}_i(\vec{\gamma}), \vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3\right).$$

In the notation of the lemma, this means that $|W| = 2^\ell \cdot p^3$. $\mathcal{B}$'s task is to distinguish whether $\vec{\gamma}$ is a random vector from the column space of X or a uniformly random vector modulo p.

$\mathcal{B}$ will implicitly set the values $a_1^i, \ldots, a_{n-1}^i$ modulo p_{j+1} of the matrix A_i to be equal to $\gamma_1, \ldots, \gamma_{n-1}$. It then provides $\mathcal{A}$ with $f_i(A_i, \text{SK}_{i-1}) = \tilde{f}_i(\vec{\gamma})$. It can compute SK$_i$ by using its knowledge of SK$_{i-1}$, the values $b_1^i, \ldots, b_{n-1}^i$ for A_i, the values $a_1^i, \ldots, a_{n-1}^i$ modulo p_k for $k \neq j+1$, and the values $\vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3$ modulo p_{j+1}. We note that $\mathcal{B}$ will not fully know

A_i, but it will know SK_i (allowing it to produce the signature here and to continue producing signatures for the rest of the simulation because it will fully know all of the subsequent update matrices).

To choose the update matrix A_{i+1}, $\mathcal{B}$ chooses $a_1^{i+1}, \ldots, a_{n-1}^{i+1}$ uniformly at random modulo N, and chooses the values $b_1^{i+1}, \ldots, b_{n-1}^{i+1}$ modulo prime factors of N not equal to p_{j+1} such that $\vec{b}_{i+1}$ satisfies the dot product conditions 1., 2., and 3. modulo primes p_k for $k \leq j$, and is random for primes p_k for $k > j+1$. Modulo p_{j+1}, $\mathcal{B}$ sets $\vec{b}_{i+1}$ as follows. Now, $\vec{t_1}, \vec{t_2}, \vec{t_3}$ span a 3-dimensional space in $\mathbb{Z}_p^{n-1}$ (recall they were chosen to be linearly independent), while the space of vectors orthogonal to both $\vec{a}_{i-2}$ and $\vec{a}_{i-1}$ has dimension $n-3$. Thus, with all but negligible probability, there exists some vector in the intersection of these spaces which is *not* orthogonal to γ, and since $\mathcal{B}$ knows the values $\vec{\gamma} \cdot \vec{t_1}, \vec{\gamma} \cdot \vec{t_2}, \vec{\gamma} \cdot \vec{t_3}$, $\mathcal{B}$ can compute a vector $\vec{b'}$ which is orthogonal to both $\vec{a}_{i-2}$ and $\vec{a}_{i-1}$ and has $\vec{\gamma} \cdot \vec{b'} = -1$. Now, the space of vectors $\in \mathbb{Z}_p^{n-1}$ which are orthogonal to the column space of X is a (uniformly random) space of dimension 3. $\mathcal{B}$ samples the vector $\vec{\delta}$ modulo p randomly from the intersection of this with the $n-3$ dimensional space of vectors which are orthogonal to $\vec{a}_{i-2}$ and $\vec{a}_{i-1}$ (note that we are working in the $(n-1)$-dimensional space $\mathbb{Z}_p^{n-1}$, so this intersection will be non-trivial). It sets $\vec{b}_{i+1} = \vec{b'} + \vec{\delta}$. This ensures that $\vec{b}_{i+1}$ satisfies conditions 1. and 2. modulo p.

For the remaining updates, $\mathcal{B}$ chooses the update matrix according to the distribution specified in the update algorithm. If $\vec{\gamma}$ is a uniformly random vector modulo p, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j,2}$. To see this, note that $\vec{b}_{i+1}$ is distributed as a random vector up to satisfying conditions 1. and 2. modulo p, since the space X is chosen randomly.

If $\vec{\gamma}$ is a random vector from the column space of X, then $\mathcal{B}$ has properly simulated $\mathrm{GameAlt}_{i,j+1}$. To see this, note that $\vec{b}_{i+1}$ is orthogonal to both $\vec{a}_{i-2}$ and $\vec{a}_{i-1} = \vec{\gamma}$ modulo p, and $\vec{b}_{i+1} \cdot \vec{a}_i = \vec{b}_{i+1} \cdot \vec{\gamma} = -1$ modulo p, and so satisfies conditions 1., 2., and 3.. We further have that $\vec{b}_{i+1}$ is distributed randomly up to these conditions, because $\vec{\delta}$ is distributed as a random vector which is orthogonal to $\vec{a}_i$, $\vec{a}_{i-1}$ and $\vec{a}_{i-2}$. To see this, note that X is distributed randomly up to the constraints that it is orthogonal to $\vec{\delta}$ and contains $\vec{\gamma}$.

Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to distinguish these two distributions with non-negligible probability. This will contradict Lemma 11 as long as ϵ is negligible.

To apply the lemma, we need:

$$|W| = 2^{\ell} p^3 \leq 4(1 - 1/p) p^{n-5} \epsilon^2,$$

so it suffices to have ℓ such that

$$\ell \leq (n-8) \log(p_{j+1}) + 2 \log(\epsilon)$$

for some negligible ϵ. For simplicity, we define $\delta = -\log(\epsilon)$. We then obtain the desired result as long as

$$\ell \leq (n-8) \log(p_{j+1}) - 2\delta,$$

for any δ such that $2^{-\delta}$ is negligible. □

Combining the results of Lemmas 22, 23, and 24, we obtain Lemma 21. As an immediate consequence of Lemma 21, we obtain Lemma 18.

B Proof of Lemma 19

Proof. We suppose there exists a PPT algorithm $\mathcal{A}$ which achieves a non-negligible difference in probability of producing a Type II forgery between $\mathrm{GameAlt}_2$ and $\mathrm{GameAlt}_1$. We will create a PPT algorithm $\mathcal{B}$ which achieves non-negligible advantage against Assumption 3.

$\mathcal{B}$ is given $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It will simulate either GameAlt$_2$ or GameAlt$_1$ with $\mathcal{A}$, depending on the value of T. We will then show that with all but negligible probability, $\mathcal{B}$ can determine when $\mathcal{A}$ is producing a Type II forgery. Thus, the non-negligible difference in $\mathcal{A}$'s probability of producing a Type II forgery will allow $\mathcal{B}$ to achieve non-negligible advantage against Assumption 3.

$\mathcal{B}$ chooses random vectors $\vec{r}, \vec{t}, \vec{c}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and random values $\alpha, \beta, f_1, f_2, \delta, \gamma, \psi \in \mathbb{Z}_N$. It sets the public parameters as:

$$R := g_4,\ gR' := gg_4^{\delta},\ uR'' := g^{\alpha}g_4^{\gamma},\ hR''' := g^{\beta}g_4^{\psi}.$$

It initializes the secret key as:

$$\vec{S}_0 = g^{\vec{r}}T^{\vec{t}}(Y_2Y_3)^{\vec{c}}g_3^{\vec{x}},$$
$$\vec{U}_0 = g^{\alpha\vec{r}}T^{\alpha\vec{t}}(Y_2Y_3)^{f_1\vec{c}}g_3^{\vec{y}},$$
$$\vec{H}_0 = g^{\beta\vec{r}}T^{\beta\vec{t}}(Y_2Y_3)^{f_2\vec{c}}g_3^{\vec{z}}.$$

If T has a nonzero component in G_{p_2}, then the exponent vectors of the G_{p_2} parts here are distributed as random vectors from the two-dimensional subspace spanned by $\vec{c}$ and $\vec{t}$ (which matches the distribution specified in GameAlt$_2$). If T does not have a G_{p_2} component, then these exponent vectors are distributed as random vectors from the one-dimensional subspace spanned by $\vec{c}$ (which matches the distribution specified in GameAlt$_1$).

When $\mathcal{A}$ makes the first signature request, $\mathcal{B}$ will choose an update matrix A_1 whose rows are orthogonal to $\vec{c}$. It will choose the values $a_1^1, \ldots, a_{n-1}^1$ for A_1 uniformly at random modulo N. We note that this first update will be properly distributed as in GameAlt$_2$ if $T \in G_{p_1p_2p_3}$, and will be properly distributed as in GameAlt$_1$ when $T \in G_{p_1p_3}$ (in this case, the G_{p_2} elements will be completely canceled out).

When $\mathcal{A}$ makes the second signature request, $\mathcal{B}$ will choose an update matrix A_2 whose rows are orthogonal to $A_1\vec{t}$. It will choose the values $a_1^2, \ldots, a_{n-1}^2$ for A_2 uniformly at random modulo N. If $T \in G_{p_1p_2p_3}$, this is a properly distributed second update for GameAlt$_2$, which cancels out the remaining G_{p_2} parts of the secret key. If $T \in G_{p_1p_3}$, this is a properly distributed second update for GameAlt$_1$, since $\vec{t}$ is random (note that no information about $\vec{t}$ is revealed by the secret key before this point, since $\vec{r}$ is uniformly random modulo p_1 and $\vec{x}, \vec{y}, \vec{z}$ are uniformly random modulo p_3).

When $\mathcal{A}$ makes the third signature request, $\mathcal{B}$ will choose a uniformly random vector $\vec{v}$ and will choose an update matrix A_3 whose rows are orthogonal to $A_2A_1\vec{v}$. It will choose the values $a_1^3, \ldots, a_{n-1}^3$ for A_3 uniformly at random modulo N. This is a properly distributed third update for either GameAlt$_2$ or GameAlt$_1$. $\mathcal{B}$ chooses the remaining update matrices according the distribution prescribed by the update algorithm. We note that $\mathcal{B}$ knows all of the secret keys and update matrices used throughout the simulation, so it can easily provide $\mathcal{A}$ with the requested leakage and signatures. If $T \in G_{p_1p_2p_3}$, then $\mathcal{B}$ has properly simulated GameAlt$_2$. If $T \in G_{p_1p_3}$, then $\mathcal{B}$ has properly simulated GameAlt$_1$.

When $\mathcal{A}$ produces a forgery (σ_1, σ_2) on m^* (that verifies correctly), $\mathcal{B}$ tests whether it is a Type I or Type II forgery by checking if the following holds:

$$e(\sigma_1, X_1X_2) \stackrel{?}{=} e(\sigma_2, (X_1X_2)^{\alpha m^* + \beta}).$$

If this equality holds, $\mathcal{B}$ will guess that $\mathcal{A}$ has produced a Type I forgery. If the equality fails, then $\mathcal{B}$ knows that $\mathcal{A}$ has produced a Type II forgery (note that this equality can only fail for a forgery that properly verifies when there is some G_{p_2} part present in σ_1 and/or σ_2).

We now argue that $\mathcal{A}$ can only produce a Type II forgery that $\mathcal{B}$ misclassifies as a Type I forgery with negligible probability. To fool $\mathcal{B}$, $\mathcal{A}$ must produce G_{p_2} parts for σ_1 and σ_2 of the form g_2^s, $g_2^{s(\alpha m^*+\beta)}$, where g_2 is a generator of G_{p_2} and s is arbitrary. This implies that $\mathcal{A}$ must be able to implicitly determine the value $\alpha m^* + \beta$ modulo p_2.

If $T \in G_{p_1 p_3}$, the initial secret key reveals *no information* about the values of α and β modulo p_2: so these remain information-theoretically hidden from $\mathcal{A}$ throughout the entire game. Thus, $\mathcal{A}$ has only a negligible chance of determining $\alpha m^* + \beta$ modulo p_2 correctly. When $T \in G_{p_1 p_2 p_3}$, the first signature involves $\alpha m_1 + \beta$ modulo p_2 in the exponent, where m_1 is the first message that $\mathcal{A}$ asks to be signed. We note that $\alpha m + \beta$ modulo p_2 is a pairwise independent function of m, and that $\mathcal{A}$ must forge for a message $m^* \neq m_1$ modulo p_2. After the second update matrix is applied, the G_{p_2} parts of the key are canceled, so the values of α, β modulo p_2 no longer appear. Hence no other signatures will contain any information about α, β modulo p_2. $\mathcal{A}$ can obtain additional information about α, β modulo p_2 only from its first two leakage queries.

We again let X denote the random variable $\alpha || \beta$ modulo p_2. This has min-entropy $2\log(p_2)$. The information the attacker learns about X can be expressed as $F(X)$ for a single function F which produces $2\ell + \log(p_2)$ bits (2ℓ bits learned from two leakage queries and $\log(p_2)$ bits learned from $\alpha m_1 + \beta$). Thus, for $\ell \leq \frac{1}{2}(\log(p_2) - 2\delta)$, by Lemma 3, the min-entropy of X conditioned on $F(X)$ will be at least δ with probability $1 - 2^{-\delta}$ (which is all but negligible probability). In this case, the probability of an attacker determining $\alpha m^* + \beta$ modulo p_2 correctly for some $m^* \neq m_i$ modulo p_2 is at most $2^{-\delta}$, which is negligible (note that $\alpha m^* + \beta$ and $\alpha m_1 + \beta$ together would fully determine α, β modulo p_2 since m^*, m_i are known). This completes the proof that $\mathcal{B}$ will incur only negligible error in determining the forgery type of $\mathcal{A}$, and hence will achieve non-negligible advantage against Assumption 3. □

C Proof of Lemma 20

Proof. We suppose there exists a PPT algorithm $\mathcal{A}$ which achieves a non-negligible difference in probability of producing a Type II forgery between GameAlt$_1$ and GameAlt$_0$. We will create a PPT algorithm $\mathcal{B}$ which achieves non-negligible advantage against Assumption 3.

$\mathcal{B}$ is given $g, g_3, g_4, X_1 X_2, Y_2 Y_3, T$. It will simulate either GameAlt$_1$ or GameAlt$_0$ with $\mathcal{A}$, depending on the value of T. We will then show that with all but negligible probability, $\mathcal{B}$ can determine when $\mathcal{A}$ is producing a Type II forgery. Thus, the non-negligible difference in $\mathcal{A}$'s probability of producing a Type II forgery will allow $\mathcal{B}$ to achieve non-negligible advantage against Assumption 3.

$\mathcal{B}$ chooses random vectors $\vec{r}, \vec{t}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and random values $\alpha, \beta, \delta, \gamma, \psi \in \mathbb{Z}_N$. It sets the public parameters as:

$$R := g_4,\ gR' := g g_4^{\delta},\ uR'' := g^{\alpha} g_4^{\gamma},\ hR''' := g^{\beta} g_4^{\psi}.$$

It initializes the secret key as:

$$\vec{S}_0 = g^{\vec{r}} T^{\vec{t}} g_3^{\vec{x}},$$
$$\vec{U}_0 = g^{\alpha\vec{r}} T^{\alpha\vec{t}} g_3^{\vec{y}},$$
$$\vec{H}_0 = g^{\beta\vec{r}} T^{\beta\vec{t}} g_3^{\vec{z}}.$$

If $T \in G_{p_1 p_3}$, then this initial secret key has no G_{p_2} components, and is distributed as in GameAlt$_0$. If $T \in G_{p_1 p_2 p_3}$, then the G_{p_2} components here are distributed as in GameAlt$_1$: the exponent vectors of the G_{p_2} parts are random vectors chosen from the same one-dimensional subspace spanned by $\vec{t}$.

$\mathcal{B}$ chooses the first update matrix A_1 randomly up to the constraint that $\vec{t} = (t_1, \ldots, t_n)$ is orthogonal to all of the rows of A_1. If $T \in G_{p_1 p_2 p_3}$, this first update cancels out the G_{p_2} parts of the initial secret key, matching the specification of GameAlt$_1$. If $T \in G_{p_1 p_3}$, then this first update matches the specification of GameAlt_0, since $\vec{t}$ is uniformly random (note that before this, the value of $\vec{t}$ modulo p_1 is hidden by the random vector $\vec{r}$, and the value of $\vec{t}$ modulo p_3 is hidden by the random vectors $\vec{x}, \vec{y}, \vec{z}$).

For the second update, $\mathcal{B}$ chooses a random vector $\vec{w}$ and chooses A_2 so that $A_2 A_1 \vec{w}$ is the all zeros vector. For the third update, $\mathcal{B}$ chooses a random vector $\vec{v}$ and chooses A_3 so that $A_3 A_2 A_1 \vec{v}$ is the all zeros vector. It chooses the remaining updates according the distribution prescribed in the update algorithm. Since $\mathcal{B}$ knows all of the secret keys and update matrices, it can easily provide $\mathcal{A}$ with the requested leakage and signatures.

If $T \in G_{p_1 p_2 p_3}$, then $\mathcal{B}$ has properly simulated GameAlt_1. If $T \in G_{p_1 p_3}$, then $\mathcal{B}$ has properly simulated GameAlt_0. When $\mathcal{A}$ produces a forgery (σ_1, σ_2) on m^* (that verifies correctly), $\mathcal{B}$ tests whether it is a Type I or Type II forgery by checking if the following holds:

$$e(\sigma_1, X_1 X_2) \stackrel{?}{=} e(\sigma_2, (X_1 X_2)^{\alpha m^* + \beta}).$$

If this equality holds, $\mathcal{B}$ will guess that $\mathcal{A}$ has produced a Type I forgery. If the equality fails, then $\mathcal{B}$ knows that $\mathcal{A}$ has produced a Type II forgery (note that this equality can only fail for a forgery that properly verifies when there is some G_{p_2} part present in σ_1 and/or σ_2).

To fool $\mathcal{B}$ into misclassifying its forgery type, $\mathcal{A}$ must produce G_{p_2} parts for σ_1 and σ_2 of the form g_2^s, $g_2^{s(\alpha m^* + \beta)}$, where g_2 is a generator of G_{p_2} and s is arbitrary. This implies that $\mathcal{A}$ must be able to implicitly determine the value $\alpha m^* + \beta$ modulo p_2.

If $T \in G_{p_1 p_3}$, the initial secret key reveals *no information* about the values of α and β modulo p_2: so these remain information-theoretically hidden from $\mathcal{A}$ throughout the entire game. Thus, $\mathcal{A}$ has only a negligible chance of determining $\alpha m^* + \beta$ modulo p_2 correctly. When $T \in G_{p_1 p_2 p_3}$, the first update matrix A_1 will cancel out the G_{p_2} parts of the secret key, and this applied *before any* signatures are computed. Thus, none of the signatures given out reveal any information about α, β modulo p_2. Hence, $\mathcal{A}$'s only opportunity to learn anything about the values of α, β modulo p_2 is in its first leakage query.

As before, we let X denote the random variable $\alpha || \beta$ modulo p_2. This has min-entropy $2\log(p_2)$. The information the attacker learns about X can be expressed as $F(X)$ for a single function F which produces ℓ bits (ℓ bits learned from a single leakage query). Thus, for $\ell \leq \log(p_2) - 2\delta$, by Lemma 3, the min-entropy of X conditioned on $F(X)$ will be at least $\log(p_2) + \delta$ with probability $1 - 2^{-\delta}$ (which is all but negligible probability). In this case, the probability of an attacker determining $\alpha m^* + \beta$ modulo p_2 correctly for some $m^* \neq m_i$ modulo p_2 is at most $2^{-\delta}$, which is negligible (note that the entropy of X conditioned on $\alpha m^* + \beta$ for a known m^* is $\log(p_2)$). This completes the proof that $\mathcal{B}$ will incur only negligible error in determining the forgery type of $\mathcal{A}$, and hence will achieve non-negligible advantage against Assumption 3.

□

D Proof of Security for our PKE Scheme

We now prove security for our PKE scheme. Our strategy here closely follows the proof strategy for our signature scheme. There is really only one significant difference. At a few points in our signature proof, we used that the variable $X := \alpha || \beta$ modulo p_2 had sufficient min-entropy in the attacker's view to prevent the attacker from producing a forgery on a new message m^* modulo p_2 involving the value $\alpha m^* + \beta$. Since a PKE attacker's task is to *distinguish* rather than *produce*, we must replace this min-entropy argument with an argument based on statistical distance. Essentially, we will start by changing all of our ciphertexts to have uniformly random G_{p_2} components, and we will need to maintain that these components look uniformly random in the attacker's view throughout the rest of our game sequence. This is a non-trivial task, since we will sometimes be producing these components in a way which is correlated with parameters that are ephemerally present in the secret key. We will argue our ciphertexts remain statistically close to uniformly random in the G_{p_2} subgroup via yet another application of Corollary 12 (derived from the lemma of [12]).

In the real security game, which we will call Game_{Real}, the ciphertext and secret keys do not include any G_{p_2} components. We will gradually move to a game where the ciphertext and all secret keys include random G_{p_2} components. In the terminology of dual system encryption, this is to say that the ciphertext and all keys have become "semi-functional". (Semi-functional ciphertexts can be decrypted by normal keys and semi-functional keys can decrypt normal ciphertexts, but semi-functional keys cannot decrypt semi-functional ciphertexts.) Once we have arrived at a game where the ciphertext and all the secret keys have random G_{p_2} components, the secret keys have become useless for decrypting the challenge ciphertext, and proving security is relatively straightforward.

Before we define our sequence of games, we must first define three possible distributions for the G_{p_2} parts of the initial secret key. We let g_2 denote a generator for G_{p_2}.

Distribution D_{Full} We define distribution D_{Full} as follows. We choose three random vectors $\vec{c}, \vec{d}, \vec{f} \in \mathbb{Z}_N^n$. We output the following three n-tuples of elements in G_{p_2}: $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$.

Distribution D_{Alt} We define distribution D_{Alt} as follows. We choose two random vectors $\vec{c}, \vec{d} \in \mathbb{Z}_N^n$. We choose $\vec{f}$ to be a random vector in the span of $\vec{c}, \vec{d}$. We output: $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$.

Distribution D_{Min} We define distribution D_{Min} as follows. We choose a random vector $\vec{c} \in \mathbb{Z}_N^n$, and we choose $\vec{d}, \vec{f}$ randomly from the one-dimensional span of $\vec{c}$. We output: $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$.

We now define the games we will use in our hybrid argument.

Game$_0$ This game is like Game_{Real}, except that the challenge ciphertext given to the attacker has random G_{p_2} components on C_1, C_2, C_3 (this is true for both encryptions of 0 and 1).

Game$'_0$ This game is like Game_0, except that the second update matrix is chosen so that a new random vector is in the kernel of the product of the first two update matrices. More precisely, we let A_1 denote the first update matrix, and $\vec{t}$ denote a randomly chosen vector. Then, A_2 is chosen randomly up to the constraint that $A_2 A_1 \vec{t}$ is the all zeros vector. The remaining updates are chosen from the distribution specified in the update algorithm.

Game$''_0$ This game is like Game'_0, except that the third update matrix is now also chosen so that a new random vector is in the kernel of the product of the first two update matrices. More precisely, we let A_1, A_2 denote the first two update matrices, and $\vec{w}$ denote a randomly chosen vector. Then, A_3 is chosen randomly up to the constraint that $A_3 A_2 A_1 \vec{w}$ is the all zeros vector. The remaining updates are chosen from the distribution specified in the update algorithm.

Game$_{Min}$ This game is like Game''_0, except that the initial key has G_{p_2} components distributed according to distribution D_{Min}, and these components are canceled out by the first update matrix (so SK_1 has no G_{p_2} components). The second and third update matrices are chosen so that new random vectors are included in the matrix product, as in Game''_0. The remaining update matrices are chosen from the distribution specified in the update algorithm.

Game$_i$ For each i from 3 to $q+3$, we define Game$_i$ as follows. The key generation is performed as described in the key generation algorithm, except that the secret key is initialized to have G_{p_2} parts which are distributed according to distribution D_{Full}. We let $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$ denote the G_{p_2} parts of the initial secret key (for $\vec{S}_0, \vec{U}_0, \vec{H}_0$ respectively). The first $i-3$ update matrices are chosen according to the distributed specified in the update algorithm, and we let $A = A_{i-3} \cdots A_1$ denote the product of these update matrices (if $i = 3$, A is the identity matrix). For the $i-2$ update, the challenger will choose a random vector $\vec{t}$ in the span of $\vec{c}, \vec{d}, \vec{f}$ and will choose the update matrix A_{i-2} randomly up to the constraint that $A_{i-2}A\vec{t}$ is the all zeros vector. We let $\vec{w}, \vec{v}$ denote random vectors such that the span of $\vec{w}, \vec{v}, \vec{t}$ is equal to the span of $\vec{c}, \vec{d}, \vec{f}$. For the $i-1$ update, the challenger will choose the update matrix A_{i-1} randomly up to the constraint that $A_{i-1}A_{i-2}A\vec{w}$ is the all zeros vector. It will then choose the update matrix A_i for the i^{th} update randomly up to the constraint that $A_iA_{i-1}A_{i-2}A\vec{v}$ is the all zeros vector. This will cancel out the G_{p_2} parts of the secret key. The rest of the updates are chosen according to the distribution specified in the update algorithm. The challenge ciphertext given to the attacker has random G_{p_2} components on C_1, C_2, and C_3 (regardless of which bit is being encrypted). We note that in Game$_{q+3}$, the challenge ciphertext and all of the secret keys the attacker receives leakage for have G_{p_2} components, and all of the update matrices are chosen according to the prescribed distribution.

GameAlt$_i$ For each i from 2 to $q+2$, we define GameAlt$_i$ as follows. The key generation is performed as described in the key generation algorithm, except that the secret key is initialized to have G_{p_2} parts which are distributed according to distribution D_{Alt}. We let $g_2^{\vec{c}}$, $g_2^{\vec{d}}$, and $g_2^{\vec{f}}$ denote the G_{p_2} parts of the initial secret key (we note that these three vectors span a two-dimensional space). The first $i-2$ update matrices are chosen properly from the distribution specified in the update algorithm. We let A denote the product of these update matrices. We let $\vec{w}, \vec{v}$ denote a random basis for the two-dimensional space spanned by $\vec{c}, \vec{d}, \vec{f}$. The update matrix A_{i-1} is chosen randomly up to the constraint that $A_{i-1}A\vec{w}$ is the all zeros vector. The update matrix A_i is chosen randomly up to the constraint that $A_iA_{i-1}A\vec{v}$ is the all zeros vector. This will cancel out the G_{p_2} parts of the secret key. For the $i+1$ update, the challenger chooses a random vector $\vec{t}$ and chooses the update matrix A_{i+1} randomly up to the constraint that $A_{i+1}A_iA_{i-1}A\vec{t}$ is the all zeros vector. The rest of the updates are chosen according to the distribution specified in the update algorithm. The challenge ciphertext given to the attacker has random G_{p_2} components on C_1, C_2, and C_3.

Game$_{Final}$ This game is like Game$_{q+3}$ in that all of the secret keys and the ciphertext elements have G_{p_2} components, and all updates are chosen according to the proper distribution. The difference is that in this game, the distribution of the ciphertext elements in the G_{p_1} subgroups is always random as well - independently of the message bit. The makes the attacker's view independent of the message bit, and so the attacker has advantage equal to zero in this game.

Using our computational assumptions as well as the lemmas developed in Subsection 5.5, we will prove that a PPT attacker's advantage changes only negligible as we move from Game$_{Real}$ to Game$_0$, then to Game$'_0$, then to Game$''_0$, then to Game$_{Min}$, then to GameAlt$_2$, then to Game$_3$, then to GameAlt$_3$, and so on, finally ending with a transition from Game$_{q+3}$ to Game$_{Final}$.

Remark 25. *We could avoid the need for Game$'_0$ and Game$''_0$ by defining our construction to choose its first three update matrices so that a new random vector is added to the kernel of the* *product of the update matrices each time. However, from the perspective of construction design, this requirement is quite unnatural, and we prefer to handle it as part of our proof.*

D.1 Transition from Game_{Real} to Game_0

We first show:

Lemma 26. *Under Assumption 2, no PPT attacker achieves a non-negligible difference in advantage between $Game_{Real}$ and $Game_0$.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which achieves a non-negligible difference in advantage between Game_{Real} and Game_0. We will create a PPT algorithm $\mathcal{B}$ which achieves a non-negligible advantage in breaking Assumption 2. $\mathcal{B}$ receives g, g_3, g_4, T, where T is either in G_{p_1} or in $G_{p_1p_2}$. $\mathcal{B}$ will simulate either Game_{Real} or Game_0 with $\mathcal{A}$, depending on the value of T.

$\mathcal{B}$ chooses $R, R', R'', R''' \in G_{p_4}$ randomly (which it can obtain by raising g_4 to randomly chosen exponents modulo N), and chooses $\vec{\alpha} \in \mathbb{Z}_N^3$ randomly. It gives the public key $\text{PK} = \{N, G, R, g^{\alpha_1}R', g^{\alpha_2}R'', g^{\alpha_3}R'''\}$ to $\mathcal{A}$. It chooses a random vector $\vec{\eta} \in \mathbb{Z}_N^3$ such that $\vec{\eta} \cdot \vec{\alpha} = 0$ modulo N, and chooses random vectors $\vec{r}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$. It initializes the secret key as $\text{SK}_0 = \{g^{\eta_1\vec{r}}g_3^{\vec{x}}, g^{\eta_2\vec{r}}g_3^{\vec{y}}, g^{\eta_2\vec{r}}g_3^{\vec{z}}\}$. This public key and secret key are properly distributed. $\mathcal{B}$ chooses all of its update matrices from the distribution specified in the update algorithm. $\mathcal{B}$ can easily respond to all leakage queries because it knows the initial secret key and all of the update matrices.

$\mathcal{B}$ produces the challenge ciphertext as follows. $\mathcal{B}$ chooses three random exponents $s, t, v \in \mathbb{Z}_N$. If the bit to be encrypted is 1, $\mathcal{B}$ sets $C_1 = T^s g_4^s, C_2 = T^t g_4^t, C_3 = T^v g_4^v$. If $T \in G_{p_1}$, these will be three uniformly random elements of $G_{p_1p_4}$ (as required for Game_{Real}). If $T \in G_{p_1p_2}$, these will be three uniformly random elements of $G_{p_1p_2p_4}$ (as required for Game_0). If the bit to be encrypted is 0, $\mathcal{B}$ sets $C_1 = T^{\alpha_1 s}g_4^s, C_2 = T^{\alpha_2 s}g_4^t, C_3 = T^{\alpha_3 s}g_4^v$. If $T \in G_{p_1}$, this is distributed as in Game_{Real}. If $T \in G_{p_1p_2}$, then the G_{p_2} parts here are random (since $\alpha_1, \alpha_2, \alpha_3$ are random modulo p_2 and uncorrelated from their values modulo p_1 which appear in the public key), so this is distributed as in Game_0. Hence, when $T \in G_{p_1}$, $\mathcal{B}$ has properly simulated Game_{Real}, and when $T \in G_{p_1p_2}$, $\mathcal{B}$ has properly simulated Game_0. $\mathcal{B}$ can therefore use the output of $\mathcal{A}$ to obtain a non-negligible advantage against Assumption 2. □

D.2 Transition from Game_0 to Game'_0

For this game transition, we will rely on Corollaries 12 and 14. As in our security proof for signatures, we will use a hybrid here over the primes dividing N. To do this, we define the following additional games:

$\text{Game}_{0,1}$ This game is like Game_0, except that the second update matrix is chosen as in Game'_0 modulo p_1 and chosen as in Game_0 for the other primes.

$\text{Game}_{0,2}$ This game is like $\text{Game}_{0,1}$, except that the second update matrix is chosen as in Game'_0 modulo p_2 as well as modulo p_1.

$\text{Game}_{0,3}$ This game is like $\text{Game}_{0,2}$, except that the second update matrix is chosen as in Game'_0 modulo p_3 as well as modulo p_1, p_2.

We let $\text{Game}_{0,0}$ be another name for Game_0 and $\text{Game}_{0,4}$ be another name for Game'_0 (this is for convenience of notation in the following lemma). We now show:

Lemma 27. *For $\ell \leq (n-6)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker can distinguish between $Game_{0,j}$ and $Game_{0,j+1}$ with non-negligible advantage for each j from 0 to 3.*

Proof. We suppose that for some such j, $\mathcal{A}$ is a PPT attacker who can distinguish between these games with non-negligible advantage. We will then create a algorithm $\mathcal{B}$ which distinguishes between the distributions $\left(\vec{\delta}, F(\vec{\tau})\right)$ and $\left(\vec{\delta}, F(\vec{\tau'})\right)$ of Corollary 12 with non-negligible advantage. Since our value of ϵ will be negligible, we will obtain a contradiction.

$\mathcal{B}$ chooses a bilinear group of order $N = p_1p_2p_3p_4$ and follows the KeyGen algorithm to produce PK and SK_0. (We note that $\mathcal{B}$ knows the primes $p_1, \ldots, p_4$ as well as generators for each prime order subgroup.) It gives PK to $\mathcal{A}$. $\mathcal{A}$ then defines the leakage function f_1 to be applied to SK_0 and the first update matrix A_1. $\mathcal{B}$ chooses A_1 as follows. It sets the vector $\vec{b}_1$ for the last column of A_1 randomly modulo N, and chooses the values for the entries of the vector $\vec{a}_1$ for the last row of A_1 randomly modulo primes not equal to p_{j+1}. The only remaining variables are the values for $\vec{a}_1$ modulo p_{j+1}.

We let $\tilde{f}_1$ denote the function of these variables modulo p_{j+1} obtained from f_1 now that all other values in SK_0, A_1 are fixed. We let $\vec{t}_1, \vec{t}_2, \vec{t}_3$ denote nonzero vectors which include the nonzero exponent vectors of the current secret key modulo p_{j+1} in the $G_{p_{j+1}}$ subgroup. We define $F : \mathbb{Z}_{p_{j+1}}^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_{p_{j+1}}^3$ as follows:

$$F(\vec{\gamma}) := \left(\tilde{f}_1(\vec{\gamma}), \vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3\right).$$

$\mathcal{B}$ obtains a sample $\left(\vec{\delta}, F(\vec{\gamma})\right)$ from one of the distributions in Corollary 12 (applied with $m = n-1$ and $p = p_{j+1}$). The task of $\mathcal{B}$ is to distinguish whether $\vec{\delta} \cdot \vec{\gamma}$ is 0 modulo p_{j+1}.

$\mathcal{B}$ implicitly sets the values of $\vec{a}_1$ equal to $\vec{\gamma}$ modulo p_{j+1}. It can then provide $f_1(\mathrm{SK}_0, A_1) = \tilde{f}_1(\vec{\gamma})$ to $\mathcal{A}$, and compute SK_1 (note that $\mathcal{B}$ will fully know SK_1, since it is given $\vec{\gamma} \cdot \vec{t}_1$, $\vec{\gamma} \cdot \vec{t}_2$, and $\vec{\gamma} \cdot \vec{t}_3$).

Next, $\mathcal{A}$ submits a second leakage function f_2. $\mathcal{B}$ chooses the update matrix A_2 as follows. With all but negligible probability, $\vec{\gamma} \cdot \vec{t}_1$ is nonzero modulo p_{j+1} (since $\vec{t}_1$ is nonzero). Thus, by multiplying $\vec{t}_1$ by a an appropriate constant, $\mathcal{B}$ can produce a vector $\vec{b'}$ which satisfies $\vec{\gamma} \cdot \vec{b'} = -1$ modulo p_{j+1}. It chooses the vector $\vec{a}_2$ for the last row randomly, and chooses $\vec{b}_2$ for the last column to be random modulo primes p_k for $k > j+1$, to satisfy $\vec{b}_2 \cdot \vec{a}_1 = -1$ modulo primes p_k for $k < j+1$, and equal to $\vec{\delta} + \vec{b'}$ modulo p_{j+1}. We note that $\mathcal{B}$ knows A_2 and SK_1, so it can easily compute $f_2(\mathrm{SK}_1, A_2)$. For the rest of the leakage requests, $\mathcal{B}$ chooses the update matrices according to the distribution prescribed in the update algorithm. $\mathcal{B}$ can easily form a properly distributed ciphertext (with random components in G_{p_2}) because it knows generators for each prime order subgroup of G.

We note by Corollary 14 that A_2 is distributed modulo each prime p_k as in Game'_0 when $\vec{b}_2$ is distributed as a random vector satisfying $\vec{b}_2 \cdot \vec{a}_1 = -1$ modulo p_k. So if $\vec{\delta}$ and $\vec{\gamma}$ are distributed as uniformly random vectors, then $\mathcal{B}$ has properly simulated $\mathrm{Game}_{0,j}$. If $\vec{\gamma}$ and $\vec{\delta}$ are distributed as random vectors up to the constraint that $\vec{\gamma} \cdot \vec{\delta} = 0$ modulo p_{j+1}, then $\vec{b}_2$ is distributed randomly modulo p_{j+1} up to the constraint that $\vec{b}_2 \cdot \vec{a}_1 = -1$, and so $\mathcal{B}$ has properly simulated $\mathrm{Game}_{0,j+1}$.

To apply the corollary, we need (setting $p = p_{j+1}$):

$$|W| = 2^\ell p^3 \leq 4(1 - 1/p)p^{n-3}\epsilon^2,$$

so we must have: $\ell \leq (n-6)\log(p) + 2\log(\epsilon)$ for some negligible ϵ. We define $\delta = -\log(\epsilon)$. Then, as long as $\ell \leq (n-6)\log(p) - 2\delta$, we have obtained a contradiction. □

As an immediate consequence, we conclude:

Lemma 28. *For $\ell \leq (n-6)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker can distinguish between Game$_0$ and Game$'_0$ with non-negligible advantage.*

D.3 Transition from Game$'_0$ to Game$''_0$

As in the proof for our signature scheme, we will employ a hybrid argument here over the primes dividing N. To do this, we define the following additional games:

Game$'_{0,1}$ This game is like Game$'_0$, except that the third update matrix is chosen so that there is a new random vector in the kernel of the matrix product modulo p_1. In other words, the third update matrix is chosen as in Game$''_0$ modulo p_1 and chosen as in Game$'_0$ modulo the other primes.

Game$'_{0,2}$ This game is like Game$'_{0,1}$, except that the third update matrix is now chosen as in Game$''_0$ modulo p_1 and p_2, and chosen as in Game$'_0$ modulo p_3, p_4.

Game$'_{0,3}$ This game is like Game$'_{0,2}$, except that the third update matrix is now chosen as in Game$''_0$ modulo p_1, p_2, p_3, and chosen as in Game$'_0$ modulo p_4.

For notational convenience, we let Game$'_{0,0}$ be another name for Game$'_0$ and let Game$'_{0,4}$ be another name for Game$''_0$. We will prove that no PPT attacker can achieve a non-negligibly different advantage between Game$'_0$ and Game$''_0$ by proving the following lemma:

Lemma 29. *When $\ell \leq (n-7)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker can distinguish between Game$'_{0,j}$ and Game$'_{0,j+1}$ with non-negligible advantage for each j from 0 to 3.*

As in our signature proof, a direct application of Lemma 11 here would result in a worse leakage than we can obtain by breaking this into a few more steps. We know from the proof of Corollary 15 that choosing A_3 as in Game$''_0$ modulo p is equivalent to choosing $\vec{b}_3$ randomly up to the constraints

1. $\vec{b}_3 \cdot \vec{a}_1 = 0$ modulo p and
2. $\vec{b}_3 \cdot \vec{a}_2 = -1$ modulo p.

We thus introduce the following intermediary game between each Game$'_{0,j}$ and Game$'_{0,j+1}$:

Game$'_{0,j,1}$ This game is like Game$'_{0,j}$, except that the third update matrix is chosen randomly modulo p_{j+1} up to satisfying condition 1. above.

We now prove:

Lemma 30. *When $\ell \leq (n-7)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker can distinguish between Game$'_{0,j}$ and Game$'_{0,j,1}$ with non-negligible advantage for each j from 0 to 3.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which distinguishes between Game$'_{0,j}$ and Game$'_{0,j,1}$ with non-negligible advantage for some j. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $\left(\vec{\delta}, F(\vec{\tau})\right)$ and $\left(\vec{\delta}, F(\vec{\tau'})\right)$ from Corollary 12 with non-negligible advantage. This will be a contradiction because we will set our parameters so that ϵ is negligible.

$\mathcal{B}$ will choose the primes p_1, p_2, p_3, p_4 as well as the bilinear group G. It will form the public key and secret key properly and will give the public key to $\mathcal{A}$. We let $\vec{t}_1$, $\vec{t}_2$, and $\vec{t}_3$ denote the current exponent vectors of the $G_{p_{j+1}}$ parts of the secret key in $\mathcal{B}$'s view. $\mathcal{B}$ also chooses a random vector $\vec{v} \in \mathbb{Z}_N^n$. $\mathcal{B}$ receives the first leakage function f_1 from $\mathcal{A}$ and will choose the first update matrix A_1 as follows. It picks $\vec{b}_1$ uniformly at random modulo N, and picks $\vec{a}_1$ uniformly at random modulo the primes not equal to p_{j+1}. The only remaining variables are the values of $\vec{a}_1$ modulo p_{j+1}. We let $\tilde{f}_1$ be the function of these values defined by f_1 with all of the other values fixed. We let $p := p_{j+1}$ and define the function $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^4$ as:

$$F(\vec{\gamma}) = \left(\tilde{f}_1(\vec{\gamma}), \vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3, \vec{\gamma} \cdot \vec{v}\right),$$

where all dot products are taken modulo p.

$\mathcal{B}$ now receives a sample $\left(\vec{\delta}, F(\vec{\gamma})\right)$ as in Corollary 12, where $m := n-1$. $\mathcal{B}$ implicitly sets $\vec{a}_1$ modulo p_{j+1} equal to $\vec{\gamma}$. It provides $\mathcal{A}$ with the leakage $f_1(\mathrm{SK}_0, A_1) = \tilde{f}_1(\vec{\gamma})$ and it can compute SK_1 because it knows $\vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3$ modulo p_{j+1}.

Now, $\mathcal{B}$ receives the second leakage function f_2 from $\mathcal{A}$. It will choose the second update matrix A_2 randomly up to the constraint that $A_2 A_1 \vec{v}$ is the all zeros vector. It can do this because it knows $\vec{\gamma} \cdot \vec{v}$ modulo p, and hence knows $A_1 \vec{v}$ modulo p_{j+1}. Since $\mathcal{B}$ fully knows A_2 and SK_1, it can compute $f_2(\mathrm{SK}_1, A_2)$ and give this to $\mathcal{A}$.

$\mathcal{B}$ receives the third leakage function f_3 from $\mathcal{A}$. It chooses A_3 as follows. It picks $\vec{a}_3$ randomly modulo N, and chooses $\vec{b}_3$ randomly modulo primes p_k for $k > j+1$. For primes p_k for $k < j+1$, it chooses $\vec{b}_3$ modulo p_k randomly up to conditions 1. and 2. above modulo p_k. It sets $\vec{b}_3$ modulo p_{j+1} equal to $\vec{\delta}$. $\mathcal{B}$ fully knows A_3 and SK_2, so it can easily answer the leakage query. For the remaining updates, $\mathcal{B}$ chooses the matrices properly from the prescribed distribution. $\mathcal{B}$ can easily form a properly distributed ciphertext (with random components in G_{p_2}) because it knows generators for each prime order subgroup of G.

If $\vec{\delta}, \vec{\gamma}$ are distributed as uniformly random vectors modulo p_{j+1}, then $\mathcal{B}$ has properly simulated $\mathrm{Game}'_{0,j}$. If $\vec{\delta}, \vec{\gamma}$ are distributed as random vectors satisfying $\vec{\delta} \cdot \vec{\gamma} = 0$ modulo p_{j+1}, then $\mathcal{B}$ has properly simulated $\mathrm{Game}'_{0,j,1}$. In applying the corollary, we need:

$$\ell \leq (n-7)\log(p_{j+1}) + 2\log(\epsilon),$$

for some negligible ϵ. Substituting $\delta := -\log(\epsilon)$, we can express this as:

$$\ell \leq (n-7)\log(p_{j+1}) - 2\delta$$

for δ such that $2^{-\delta}$ is negligible. □

Lemma 31. *When $\ell \leq (n-7)\log(p_{j+1}) - 2\delta$ for δ such that $2^{-\delta}$ is negligible, no PPT attacker can distinguish between $\mathrm{Game}'_{0,j,1}$ and $\mathrm{Game}'_{0,j+1}$ with non-negligible advantage for each j from 0 to 3.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which distinguishes between $\mathrm{Game}'_{0,j,1}$ and $\mathrm{Game}'_{0,j+1}$ with non-negligible advantage for some j. We will create a PPT algorithm $\mathcal{B}$ which distinguishes between the distributions $(X, F(X \cdot T))$ and $(X, F(Y))$ from Lemma 11 with non-negligible advantage. This will be a contradiction because we will set our parameters so that ϵ is negligible.

$\mathcal{B}$ will choose the primes p_1, p_2, p_3, p_4 as well as the bilinear group G. It will form the public key and secret key properly and will give the public key to $\mathcal{A}$. $\mathcal{B}$ chooses $\vec{t}_1$, $\vec{t}_2$, and $\vec{t}_3$ to be three linearly independent vectors whose span includes the current exponent vectors of the $G_{p_{j+1}}$ parts of the secret key in $\mathcal{B}$'s view. $\mathcal{B}$ receives the first leakage function f_1 from $\mathcal{A}$ and chooses $\vec{a}_1$ and $\vec{b}_1$ for A_1 uniformly at random modulo N. $\mathcal{B}$ knows SK_0 and A_1, so it can easily compute $f_1(\mathrm{SK}_0, A_1)$ and it provides this to $\mathcal{A}$.

$\mathcal{A}$ sends $\mathcal{B}$ the second leakage function, f_2. $\mathcal{B}$ chooses A_2 as follows. It chooses a new random vector $\vec{v} \in \mathbb{Z}_N^n$ and chooses $\vec{b}_2$ modulo N so that $A_2 A_1 \vec{v}$ is the all zeros vector (note that this does not constrain the values of $\vec{a}_2$ in any way). It chooses the values of $\vec{a}_2$ randomly modulo the primes not equal to p_{j+1}. This leaves the values of $\vec{a}_2$ modulo p_{j+1} as the only variables. We let $\tilde{f}_2$ denote the function of these variables obtained by considering f_2 with all the other values fixed. We let $p := p_{j+1}$ and we define $F : \mathbb{Z}_p^{n-1} \to \{0,1\}^\ell \times \mathbb{Z}_p^3$ as:

$$F(\vec{\gamma}) = \left(\tilde{f}_2(\vec{\gamma}), \vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3\right),$$

where all dot products are taken modulo p.

$\mathcal{B}$ now receives a sample $(X, F(\vec{\gamma}))$ as in Lemma 11, where $m := n-1$, $k := n-3$, and $d := 1$. It implicitly sets $\vec{a}_2 = \vec{\gamma}$ modulo p_{j+1}. It can provide $\mathcal{A}$ with $f_2(\mathrm{SK}_1, A_2) = \tilde{f}_2(\vec{\gamma})$ and it can also compute SK_2 from its knowledge of $\vec{\gamma} \cdot \vec{t}_1$, $\vec{\gamma} \cdot \vec{t}_2$, and $\vec{\gamma} \cdot \vec{t}_3$ modulo p_{j+1}.

Upon receiving the next leakage function f_3 from $\mathcal{A}$, $\mathcal{B}$ chooses A_3 as follows. It sets $\vec{a}_3$ randomly modulo N. It chooses $\vec{b}_3$ randomly modulo primes p_k where $k > j+1$, and randomly up to conditions 1. and 2. modulo primes p_k for $k < j+1$. With all but negligible probability, $\vec{\gamma} \cdot \vec{t}_1, \vec{\gamma} \cdot \vec{t}_2, \vec{\gamma} \cdot \vec{t}_3$ are all nonzero modulo p_{j+1}, and the 3-dimensional span of $\vec{t}_1, \vec{t}_2, \vec{t}_3$ non-trivially intersects the $n-2$-dimensional space of vectors in $\mathbb{Z}_p^{n-1}$ that are orthogonal to $\vec{a}_1$. This allows $\mathcal{B}$ to compute a vector $\vec{b'}$ which is orthogonal to $\vec{a}_1$ and also satisfies $\vec{b'} \cdot \vec{\gamma} = -1$. $\mathcal{B}$ now chooses a random vector $\delta \in \mathbb{Z}_p^{n-1}$ from the intersection of vectors orthogonal to X and also to $\vec{a}_1$. (With all but negligible probability, the space of vectors orthogonal to X is a 2-dimensional space in $\mathbb{Z}_p^{n-1}$ which non-trivially intersects the space of vectors orthogonal to $\vec{a}_1$ in $\mathbb{Z}_p^{n-1}$.) $\mathcal{B}$ sets $\vec{b}_3 = \vec{\delta} + \vec{b'}$. This ensures that $\vec{b}_3$ satisfies condition 1. modulo p_{j+1}. $\mathcal{B}$ knows A_3 and SK_2, so it can easily satisfy the leakage request. $\mathcal{B}$ chooses the remaining updates according to the prescribed distribution. $\mathcal{B}$ can easily form a properly distributed ciphertext (with random components in G_{p_2}) because it knows generators for each prime order subgroup of G.

If $\vec{\gamma}$ is uniformly random, then $\vec{\delta}$ is distributed as a random vector up to the constraint that $\vec{\delta} \cdot \vec{a}_1 = 0$ modulo p. Thus, $\mathcal{B}$ has properly simulated $\mathrm{Game}'_{0,j,1}$. If $\vec{\gamma}$ is distributed as a random vector from the column space of X, then $\vec{\delta}$ is distributed as a random vector up to the constraints that $\vec{\delta} \cdot \vec{\gamma} = 0$ and $\vec{\delta} \cdot \vec{a}_1 = 0$ modulo p (note that the space X is randomly distributed up to the constraints that it contains γ and is orthogonal to δ). Thus, $\mathcal{B}$ has properly simulated $\mathrm{Game}'_{0,j+1}$.

In applying the lemma, we need:

$$\ell \leq (n-7)\log(p_{j+1}) + 2\log(\epsilon)$$

for some negligible ϵ. Substituting $\delta := -\log(\epsilon)$, we can write this as:

$$\ell \leq (n-7)\log(p_{j+1}) - 2\delta$$

for δ such that $2^{-\delta}$ is negligible. □

From the combination of the previous two lemmas, we obtain Lemma 29. As an immediate consequence, we have:

Lemma 32. *When $\ell \leq (n-7)\log(p_j) - 2\delta$ for all p_j dividing N and δ such that $2^{-\delta}$ is negligible, no PPT attacker can achieve a non-negligible difference in advantage between Game$'_0$ and Game$''_0$.*

D.4 Transition from Game$''_0$ to Game$_{Min}$

Lemma 33. *Under Assumption 3, no PPT attacker can achieve a non-negligible difference in advantage between Game$''_0$ and Game$_{Min}$, for $\ell \leq \log(p_2) - 2\delta$, where δ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which achieves a non-negligible difference in advantage between Game$''_0$ and Game$_{Min}$. We will create a PPT algorithm $\mathcal{B}$ which breaks Assumption 3. $\mathcal{B}$ receives $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It chooses random elements $R, R', R'', R''' \in G_{p_4}$, and random vectors $\vec{\alpha}, \vec{\eta} \in \mathbb{Z}_N^3$ such that $\vec{\alpha} \cdot \vec{\eta} = 0$ modulo N. It sets the public key as: PK $= \{N, G, R, g^{\alpha_1}R', g^{\alpha_2}R'', g^{\alpha_3}R'''\}$. To initialize the secret key, it also chooses random vectors $\vec{r}, \vec{t}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and sets:

$$\vec{S}_0 = g^{\eta_1\vec{r}}T^{\eta_1\vec{t}}g_3^{\vec{x}},$$
$$\vec{U}_0 = g^{\eta_2\vec{r}}T^{\eta_2\vec{t}}g_3^{\vec{y}},$$
$$\vec{H}_0 = g^{\eta_3\vec{r}}T^{\eta_3\vec{t}}g_3^{\vec{z}}.$$

If $T \in G_{p_1p_3}$, this secret key has no G_{p_2} parts, and is distributed properly for Game$''_0$. If $T \in G_{p_1p_2p_3}$, this secret key has G_{p_2} parts distributed according to distribution D_{Min}, and thus is distributed properly for Game$_{Min}$.

Now, $\mathcal{B}$ chooses the first update matrix A_1 so that $A_1\vec{t}$ is the all zeros vector. This cancels out the T terms from the secret key. If T has a G_{p_2} component, this cancels the G_{p_2} components from the key, as required in Game$_{Min}$. If T does not have a G_{p_2} component, then A_1 is a properly distributed update, since no information about $\vec{t}$ is previously revealed (note that its value modulo p_1 is hidden by the random vector $\vec{r}$, and its value modulo p_3 is hidden by the random vectors $\vec{x}, \vec{y}, \vec{z}$). $\mathcal{B}$ chooses the next two update matrices so that a new random vector is in the kernel of the matrix product each time. It chooses all remaining update matrices from the distribution prescribed by the update algorithm.

$\mathcal{B}$ forms the challenge ciphertext as follows. It chooses random exponents $s, t, v \in \mathbb{Z}_N$. If it is encrypting 0, it produces three random elements of $G_{p_1p_2p_4}$ by raising $X_1X_2g_4$ to the random exponents. If it is encrypting 1, it sets:

$$C_1 = (X_1X_2)^{s\alpha_1}g_4^s,\ C_2 = (X_1X_2)^{s\alpha_2}g_4^t,\ C_3 = (X_1X_2)^{s\alpha_3}g_4^v.$$

To argue that this appears to be a properly distributed encryption of 0 in the attacker's view, we must argue that $\vec{\alpha}$ modulo p_2 is statistically close to a uniformly random vector modulo p_2 in $\mathcal{A}$'s view. We note that the public parameters reveal no information about $\vec{\alpha}$ modulo p_2, nor do any of the update matrices. The only parameters correlated to $\vec{\alpha}$ modulo p_2 are the values η_1, η_2, η_3 modulo p_2. If $T \in G_{p_1p_3}$, these never appear in the secret key at all and remain completely hidden from the attacker. If $T \in G_{p_1p_2p_3}$, the attacker gets one opportunity to obtain leakage involving $\vec{\eta}$ modulo p_2 before these values are canceled out of the secret key by the first update matrix. This leakage function must be determined *before* the attacker receives the challenge ciphertext.

The information that the attacker learns about $\vec{\eta}$ modulo p_2 can be expressed as a single function $F(\vec{\eta})$ outputting ℓ bits. Thus, by Corollary 12, as long as

$$\ell \leq \log(p_2) + 2\log(\epsilon)$$

for some negligible ϵ, the distribution of $\vec{\eta}, \vec{\alpha}$ modulo p_2 in $\mathcal{A}$'s view is statistically close to the distribution producing two uniformly random vectors. For convenience, we let $\delta := -\log(\epsilon)$. We can then state the leakage bound required here as $\ell \leq \log(p_2) - 2\delta$.

Hence, when $T \in G_{p_1p_2p_3}$, the ciphertext produced by $\mathcal{B}$ is statistically close (within negligible distance) to the distribution required in Game$_{Min}$, so $\mathcal{B}$ can use the output of $\mathcal{A}$ to attain non-negligible advantage against Assumption 3. □

D.5 Transition from Game$_{Min}$ to GameAlt$_2$

Lemma 34. *Under Assumption 3, no PPT attacker can achieve a non-negligible difference in advantage between Game$''_0$ and Game$_{Min}$, for $\ell \leq \frac{1}{2}(\log(p_2) - 2\delta)$, where δ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ with a non-negligible difference in advantage between Game$''_0$ and Game$_{Min}$. We will create a PPT algorithm which breaks Assumption 3. $\mathcal{B}$ receives $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It chooses random elements R, R', R'', R''' from G_{p_4}, and random vectors $\vec{\alpha}, \vec{\eta} \in \mathbb{Z}_N^3$ subject to the constraint that $\vec{\alpha} \cdot \vec{\eta} = 0$ modulo N. It sets the public key as PK $= \{N, G, R, g^{\alpha_1}R', g^{\alpha_2}R'', g^{\alpha_3}R'''\}$. It then chooses random vectors $\vec{r}, \vec{t}, \vec{c}, \vec{x}, \vec{y}, \vec{z}$, and two random exponents $f_1, f_2 \in \mathbb{Z}_N$. It initializes the secret key as:

$$\vec{S}_0 = g^{\eta_1 \vec{r}} T^{\eta_1 \vec{t}} (Y_2Y_3)^{\vec{c}} g_3^{\vec{x}},$$

$$\vec{U}_0 = g^{\eta_2 \vec{r}} T^{\eta_2 \vec{t}} (Y_2Y_3)^{f_1 \vec{c}} g_3^{\vec{y}},$$

$$\vec{H}_0 = g^{\eta_3 \vec{r}} T^{\eta_3 \vec{t}} (Y_2Y_3)^{f_2 \vec{c}} g_3^{\vec{z}}.$$

We note that the G_{p_1} and G_{p_3} parts here are properly distributed. If T has no G_{p_2} component, then the G_{p_2} parts here are distributed according to distribution D_{Min} (as in Game$_{Min}$), and if T has a G_{p_2} component, the G_{p_2} parts here are distributed according to distribution D_{Alt} (as in GameAlt$_2$).

The first update matrix A_1 is chosen so that $A_1\vec{c}$ is the all zeros vector. This cancels out all of the Y_2Y_3 terms. If T has no G_{p_2} component, then this cancels all of the G_{p_2} terms from the secret key. If T has a G_{p_2} component, we let g_2^{τ} denote the G_{p_2} part of T, and g_2^{y} denote Y_2. Then $\vec{t}$ is distributed as a random vector in the 2-dimensional span of $\eta_1\tau\vec{t} + y\vec{c}$, $\eta_2\tau\vec{t} + f_1y\vec{c}$, and $\eta_3\tau\vec{t} + f_2y\vec{c}$. This means the first update will be distributed as in GameAlt$_2$ in this case.

The second update matrix A_2 is chosen so that $A_2A_1\vec{t}$ is the all zeros vector. If T has no G_{p_2} component, then $\vec{t}$ is a uniformly random vector here, since its value before this has been hidden modulo p_1 by the random vector $\vec{r}$ and hidden modulo p_3 by the random vectors $\vec{x}, \vec{y}, \vec{z}$. In this case, the second update is distributed as in Game$_{Min}$. If T has a G_{p_2} component, this update cancels out the remaining G_{p_2} parts of the secret key, and is distributed as in GameAlt$_2$.

The third update matrix A_3 is chosen so that a new random vector is in the kernel of $A_3A_2A_1$. The remaining updates are chosen from the distribution prescribed by the update algorithm. These are properly distributed for both Game$_{Min}$ and GameAlt$_2$.

The challenge ciphertext is made exactly as in Lemma 33. Again we must argue that $\vec{\alpha}$ modulo p_2 is statistically close a uniformly random vector modulo p_2. The public key reveals no information about $\vec{\alpha}$ modulo p_2, nor do any of the update matrices. The attacker's only opportunity to learn information about $\vec{\alpha}$ modulo p_2 is through leakage on the correlated vector $\vec{\eta}$ modulo p_2, which appears in the secret key when $T \in G_{p_1p_2p_3}$ until it is canceled out by the second update matrix. Thus, the attacker learns only 2ℓ bits of information about $\vec{\alpha}$.

Thus, by Corollary 12, as long as

$$2\ell \leq \log(p_2) + 2\log(\epsilon)$$

for some negligible ϵ, the distribution of $\vec{\eta}, \vec{\alpha}$ modulo p_2 in $\mathcal{A}$'s view is statistically close to the distribution producing two uniformly random vectors. For convenience, we let $\delta := -\log(\epsilon)$. We can then state the leakage bound required here as $\ell \leq \frac{1}{2}(\log(p_2) - 2\delta)$.

Hence, when $T \in G_{p_1p_2p_3}$, the ciphertext produced by $\mathcal{B}$ is statistically close (within negligible distance) to the distribution required in GameAlt$_2$. When $T \in G_{p_1p_3}$, $\vec{\alpha}$ modulo p_2 is uniformly random in the attacker's view, and so the challenge ciphertext is properly distributed as in Game$_{Min}$. Therefore, $\mathcal{B}$ can use the output of $\mathcal{A}$ to attain non-negligible advantage against Assumption 3. □

D.6 Transition from GameAlt$_i$ to Game$_{i+1}$

Lemma 35. *Under Assumption 3, no PPT attacker can achieve a non-negligible difference in advantage between GameAlt$_i$ and Game$_{i+1}$, for each i from 2 to $q+2$ for $\ell \leq \frac{1}{3}(\log(p_2) - 2\delta)$, where the parameter δ is chosen so that $2^{-\delta}$ is negligible.*

Proof. This is very similar to the proof of Lemma 9. We suppose there exists a PPT attacker $\mathcal{A}$ which achieves a non-negligible difference in advantage between GameAlt$_i$ and Game$_{i+1}$, for some fixed i. We will create a PPT algorithm $\mathcal{B}$ which breaks Assumption 3. $\mathcal{B}$ receives $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. It simulates either GameAlt$_i$ or Game$_{i+1}$ with $\mathcal{A}$, depending on the value of T.

$\mathcal{B}$ chooses random elements $R, R', R'', R''' \in G_{p_4}$, and random vectors $\vec{\alpha}, \vec{\eta} \in \mathbb{Z}_N^3$ such that $\vec{\alpha} \cdot \vec{\eta} = 0$ modulo N. It also chooses random vectors $\vec{r}, \vec{t}, \vec{c}, \vec{d}, \vec{x}, \vec{y}, \vec{z} \in \mathbb{Z}_N^n$ and random values $f_1, f_2, \in \mathbb{Z}_N$. It sets the public key as PK $= \{N, G, R, g^{\alpha_1}R', g^{\alpha_2}R'', g^{\alpha_3}R'''\}$ and gives this to $\mathcal{A}$. It initializes the secret key as:

$$\vec{S}_0 := g^{\eta_1\vec{r}}T^{\eta_1\vec{t}}(Y_2Y_3)^{\vec{c}}g_3^{\vec{x}},$$

$$\vec{U}_0 := g^{\eta_2\vec{r}}T^{\eta_2\vec{t}}(Y_2Y_3)^{\vec{d}}g_3^{\vec{y}},$$

$$\vec{H}_0 := g^{\eta_3\vec{r}}T^{\eta_3\vec{t}}(Y_2Y_3)^{f_1\vec{c}+f_2\vec{d}}g_3^{\vec{z}}.$$

We note that the G_{p_1} and G_{p_3} parts here are properly distributed, and if $T \in G_{p_1p_3}$, then the G_{p_2} parts are distributed according to distribution D_{Alt}, and if $T \in G_{p_1p_2p_3}$, the G_{p_2} parts are distributed according to distribution D_{Full}.

The update matrices are chosen exactly as in the proof of Lemma 9. The challenge ciphertext is constructed as follows. If $\mathcal{B}$ is encrypting a 1, it can produce three uniformly random elements of $G_{p_1p_2p_4}$ by raising $(X_1X_2)g_4$ to three random exponents modulo N. This will be a properly distributed encryption of 1 in either GameAlt$_i$ or Game$_{i+1}$. If $\mathcal{B}$ is encrypting a 0, it will choose three random elements W, W', W'' from G_{p_4} and a random exponent $s \in \mathbb{Z}_N$ and produce the ciphertext:

$$C_1 = (X_1X_2)^{\alpha_1 s}W,\ C_2 = (X_1X_2)^{\alpha_2 s}W',\ C_3 = (X_1X_2)^{\alpha_3 s}W''.$$

Now, we must argue that in the attacker's view, the distribution of the G_{p_2} parts of this ciphertext is statistically close to the uniform distribution over $G_{p_2}^3$. Equivalently, we must show that the vector $(\alpha_1, \alpha_2, \alpha_3)$ modulo p_2 is statistically close to a uniformly random vector in $\mathbb{Z}_{p_2}^3$. We will rely on Corollary 12 applied with $p := p_2$ and $m := 3$. We note that the public key reveals no information about the values $\alpha_1, \alpha_2, \alpha_3$ modulo p_2. If $T \in G_{p_1p_3}$, then the values of η_1, η_2, η_3 modulo p_2 are never involved in the secret keys at all, and these are the only values related to $\alpha_1, \alpha_2, \alpha_3$ in any way. Hence $\vec{\alpha}$ is distributed as a random vector modulo p_2 in this case.

If $T \in G_{p_1p_2p_3}$, then we let g_2^τ denote the G_{p_2} part of T, and define y by $g_2^y = Y_2$. Now, the initial exponent vectors of the G_{p_2} parts of the key, namely $\tau\eta_1\vec{t} + y\vec{c}$, $\tau\eta_2\vec{t} + y\vec{d}$, and

$\tau\eta_3\vec{t} + f_1\vec{c} + f_2\vec{d}$ and distributed as three uniformly random vectors, and reveal no information about η_1, η_2, η_3 modulo p_2. Since the first $i-2$ update matrices are chosen randomly according to the proper distribution, no information about η_1, η_2, η_3 modulo p_2 can be learned until the $i-1$ update. This gives $\mathcal{A}$ a limited window of three leakage queries (and hence 3ℓ bits) to learn information about $\vec{\eta}$ modulo p_2 before these values are completely canceled out of the secret key. We note that these leakage functions must be specified *before* $\mathcal{A}$ sees the challenge ciphertext. Now, by Corollary 12, as long as

$$3\ell \leq \log(p_2) + 2\log(\epsilon)$$

for some negligible ϵ, the distribution of $\vec{\eta}, \vec{\alpha}$ modulo p_2 in $\mathcal{A}$'s view is statistically close to the distribution producing two uniformly random vectors. For convenience, we let $\delta := -\log(\epsilon)$. We can then state the leakage bound required here as $\ell \leq \frac{1}{3}(\log(p_2) - 2\delta)$.

In summary, when $T \in G_{p_1p_3}$, $\mathcal{B}$ has properly simulated GameAlt$_i$. When $T \in G_{p_1p_2p_3}$, $\mathcal{B}$ has produced a simulation which is statistically close (within negligible distance) of a proper simulation of Game$_{i+1}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break Assumption 3 with non-negligible advantage. □

D.7 Transition from Game$_i$ to GameAlt$_i$

The proof for this transition is very similar to the proof of the analogous transition for our signature scheme. We begin by defining two additional games (these are directly analogous to our game definitions in the signature proof):

GameAlt$'_i$ This game is like GameAlt$_i$ except that the update matrix for the $i+1$ update is now chosen according to the distribution specified in the update algorithm.

GameAlt$''_i$ This game is like GameAlt$'_i$ except that the update matrix for the $i-2$ update is now chosen so that a new random vector is in the kernel of the product of the update matrices applied (up to and including the $i-2$ update). We note that for $i=3$, this is the same as GameAlt$'_3$, since the first matrix is also chosen to be rank $n-1$.

We will prove that any PPT attacker's advantage changes only negligibly between Game$_i$ and GameAlt$_i$ by showing that it changes only negligibly between Game$_i$ and GameAlt$''_i$, between GameAlt$''_i$ and GameAlt$'_i$, and finally between GameAlt$'_i$ and GameAlt$_i$.

We now prove:

Lemma 36. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s advantage between Game$_i$ and GameAlt$''_i$ is negligible as long as $\ell \leq \log(p_2) - 2\delta$, for each i from 3 to $q+2$. Here, $\delta > 0$ is a parameter chosen so that $2^{-\delta}$ is negligible.*

Proof. This is very similar to the proof of Lemma 10. We suppose there exists a PPT attacker $\mathcal{A}$ with a non-negligible difference in advantage between Game$_i$ and GameAlt$''_i$ for some i. We will create a PPT algorithm $\mathcal{B}$ which breaks Assumption 3. $\mathcal{B}$ is given $g, g_3, g_4, X_1X_2, Y_2Y_3, T$. $\mathcal{B}$ forms the public key and initial secret key exactly as in the proof of Lemma 35. It chooses the update matrices exactly as in the proof of Lemma 10. It produces the challenge ciphertext as in the proof of Lemma 35.

Again, we must argue that encryptions of 0 have G_{p_2} components which are statistically close to uniform in $\mathcal{A}$'s view - i.e. that $\vec{\alpha}$ modulo p_2 is statistically close to a uniformly random vector modulo p_2. When T has no G_{p_2} component, $\vec{\alpha}$ is uniformly random modulo p_2 in $\mathcal{A}$'s view, as required. When T has a G_{p_2} component, the attacker's only opportunity to learn about $\vec{\alpha}$ comes from the $i-2$ update, where the T terms are canceled out of the secret key. Before this, the initial exponent vectors for the G_{p_2} components of the secret key and the independently random update matrices reveal no information about $\vec{\eta}$ modulo p_2. After this, the terms involving $\vec{\eta}$ modulo p_2 have been canceled out. Thus, the attacker learns only ℓ bits of information which are related to $\vec{\eta}$ modulo p_2, and this leakage function is specified before the challenge ciphertext is given to the attacker. Thus, by Corollary 12, as long as

$$\ell \leq \log(p_2) + 2\log(\epsilon)$$

for some negligible ϵ, the distribution of the G_{p_2} parts of the challenge ciphertext will be within negligible statistical distance of the uniform distribution on $G_{p_2}^3$. We let $\delta := -\log(\epsilon)$ and express our leakage bound as $\ell \leq \log(p_2) - 2\delta$.

In summary, when $T \in G_{p_1p_3}$, $\mathcal{B}$ has properly simulated GameAlt$''_i$. When $T \in G_{p_1p_2p_3}$, $\mathcal{B}$ has produced a simulation which is statistically close (within negligible distance) of a proper simulation of Game$_i$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break Assumption 3 with non-negligible advantage. □

Lemma 37. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s advantage between GameAlt$''_i$ and GameAlt$'_i$ for each i from 3 to $q+2$ is negligible as long as $\ell \leq (n-8)\log(p_j) - 2\delta$ for all primes p_j dividing N, where $\delta > 0$ is a parameter such that $2^{-\delta}$ is negligible.*

Proof. This follows from the same proof as the proof of Lemma 17, except that the setup for the G_{p_1} elements now follows the PKE setup and $\mathcal{B}$ will produce a challenge ciphertext instead of signatures. Note that it can easily produce a properly distributed challenge ciphertext because it knows all of the primes p_1, p_2, p_3, p_4 and generators for each corresponding prime order subgroup of G. □

Lemma 38. *Under Assumption 3, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s advantage between GameAlt$'_i$ and GameAlt$_i$ for each i from 3 to $q+2$ is negligible as long as $\ell \leq (n-8)\log(p_j) - 2\delta$ for all p_j dividing N and for δ such that $2^{-\delta}$ is negligible.*

Proof. This follows from the same proof as the proof of Lemma 18, except that the setup for the G_{p_1} elements now follows the PKE setup and $\mathcal{B}$ will produce a challenge ciphertext instead of signatures. Note that it can easily produce a properly distributed challenge ciphertext because it knows all of the primes dividing N as well as generators for each prime order subgroup of G. □

D.8 Transition from Game$_{q+3}$ to Game$_{Final}$

We finally show:

Lemma 39. *Under Assumption 1, for any polynomial time attacker $\mathcal{A}$, the difference in $\mathcal{A}$'s advantage between Game$_{q+3}$ and Game$_{Final}$ is negligible.*

Proof. We suppose that $\mathcal{A}$ is a PPT attacker which attains a non-negligible difference in advantage between Game$_{q+3}$ and Game$_{Final}$. We will create a PPT algorithm $\mathcal{B}$ which breaks Assumption 1. $\mathcal{B}$ is given $g_3, g_4, X_1X_4, Y_1Y_2Y_3, Z_2Z_3, T$. $\mathcal{B}$ chooses $\vec{\alpha}, \vec{\eta}$ randomly in $\mathbb{Z}_N^3$ up to the constraint that $\vec{\alpha} \cdot \vec{\eta} = 0$ modulo N. It chooses random elements $R, R', R'', R''' \in G_{p_4}$ and sets the public key as: PK $= \{N, G, R, (X_1X_4)^{\alpha_1}R', (X_1X_4)^{\alpha_2}R'', (X_1X_4)^{\alpha_3}R'''\}$. It chooses random vectors $\vec{r}, \vec{x}, \vec{y}, \vec{z}$ and initializes the secret key as:

$$\vec{S}_0 = (Y_1Y_2Y_3)^{\eta_1\vec{r}}(Z_2Z_3)^{\vec{x}},$$

$$\vec{U}_0 = (Y_1Y_2Y_3)^{\eta_2\vec{r}}(Z_2Z_3)^{\vec{y}},$$
$$\vec{H}_0 = (Y_1Y_2Y_3)^{\eta_3\vec{r}}(Z_2Z_3)^{\vec{z}}.$$

We note that the G_{p_1} parts here are properly distributed, and the G_{p_2} and G_{p_3} parts are uniformly random. $\mathcal{B}$ will choose all of the update matrices from the distribution specified in the update algorithm. It knows the initial secret key and all of the update matrices, so it can easily fulfill all of $\mathcal{A}$'s leakage requests.

$\mathcal{B}$ produces the challenge ciphertext as follows. If it is encrypting 1, it produces three uniformly random elements of $G_{p_1p_2p_4}$ by raising X_1X_4T to uniformly random powers modulo N (this will produce uniformly random elements of $G_{p_1p_2p_4}$ for either distribution of T). If it is encrypting 0, it chooses random exponents $s, t, v \in \mathbb{Z}_N$ and sets:

$$C_1 = (X_1X_4)^{s\alpha_1}T^s, \ C_2 = (X_1X_4)^{s\alpha_2}T^t, \ C_3 = (X_1X_4)^{s\alpha_3}T^v.$$

If $T \in G_{p_2p_4}$, this will have uniformly random components in G_{p_2} and G_{p_4}, but the G_{p_1} components will be properly distributed for an encryption of 0 in Game$_{q+3}$. If $T \in G_{p_1p_2p_4}$, then the G_{p_1} components will be uniformly random as well.

Thus, when $T \in G_{p_2p_4}$, $\mathcal{B}$ has properly simulated Game$_{q+3}$, and when $T \in G_{p_1p_2p_4}$, $\mathcal{B}$ has properly simulated Game$_{Final}$. $\mathcal{B}$ can then use the output of $\mathcal{A}$ to break Assumption 1 with non-negligible advantage. $\square$

Combining this with the results of the preceding subsections, we obtain Theorem 5.