Sequential Aggregate Signatures with Lazy Verification from Trapdoor Permutations[*]

Kyle Brogle[†] Sharon Goldberg[‡] Leonid Reyzin[§]

June 9, 2014

Abstract

Sequential aggregate signature schemes allow n signers, in order, to sign a message each, at a lower total cost than the cost of n individual signatures. We present a sequential aggregate signature scheme based on trapdoor permutations (*e.g.,* RSA). Unlike prior such proposals, our scheme does not require a signer to retrieve the keys of other signers and verify the aggregate-so-far before adding its own signature. Indeed, we do not even require a signer to *know* the public keys of other signers!

Moreover, for applications that require signers to verify the aggregate anyway, our schemes support *lazy verification*: a signer can add its own signature to an unverified aggregate and forward it along immediately, postponing verification until load permits or the necessary public keys are obtained. This is especially important for applications where signers must access a large, secure, and current cache of public keys in order to verify messages. The price we pay is that our signature grows slightly with the number of signers.

We report a technical analysis of our scheme (which is provably secure in the random oracle model), a detailed implementation-level specification, and implementation results based on RSA and OpenSSL. To evaluate the performance of our scheme, we focus on the target application of BGPsec (formerly known as Secure BGP), a protocol designed for securing the global Internet routing system. There is a particular need for lazy verification with BGPsec, since it is run on routers that must process signatures extremely quickly, while being able to access tens of thousands of public keys. We compare our scheme to the algorithms currently proposed for use in BGPsec, and find that our signatures are considerably shorter than nonaggregate RSA (with the same sign and verify times) and have an order of magnitude faster verification than nonaggregate ECDSA, although ECDSA has shorter signatures when the number of signers is small.

Keywords: Aggregate signatures, RSA, lazy verification, BGP

[*]An extended abstract of this paper appeared in [BGR12]

[†]Stanford University Department of Computer Science `broglek@stanford.edu` (work done while at Boston University)

[‡]Boston University Department of Computer Science `http://www.cs.bu.edu/fac/goldbe`

[§]Boston University Department of Computer Science `http://www.cs.bu.edu/fac/reyzin`

Contents

1 Introduction **2**
- 1.1 Aggregate signatures from trapdoor permutations 2
- 1.2 The need for lazy verification 3
- 1.3 Overview of our contributions 5

2 Preliminaries **6**

3 Our basic signature scheme **8**

4 Security proof **10**
- 4.1 Description of the reduction R 11
  - 4.1.1 Data structures used by R 11
  - 4.1.2 Algorithms used by R 12
- 4.2 Analysis of the reduction 14

5 Shorter signatures via input-dependent randomness **14**
- 5.1 Modifying the scheme 14
- 5.2 Key insight for the security proof 14

6 Implementation and Evaluation **16**

A Lazy verification and prior TDP-based proposals **22**

B Proof of Theorem 4.2 **23**
- B.1 Proof of Correct Simulation Lemma B.1 23
- B.2 Proof of Abort Probability Lemma B.2 24

C Analysis of the Scheme with Input-Dependent Randomness **26**

D Combinatorial interlude: A proof of Theorem 5.2 **27**

E Proving Security without the Claw-Free Assumption **29**

F Handling Permutations with Different Domains, such as RSA **29**

G Our Scheme and History-Free Signatures **31**

1 Introduction

Aggregate signatures schemes allow n signers to produce a digital signature that authenticates n messages, one from each signer. This can be securely accomplished by simply concatenating together n ordinary digital signatures, individually produced by each signer. An aggregate signature is designed to maintain the security of this basic approach, while having length much shorter than n individual signatures. To achieve this, many prior schemes *e.g.,* [LMRS04, Nev08] relied on a seemingly innocuous assumption; namely, that each signer needs to *verify* the aggregate signature so far, before adding its own signature on a new message. In this paper, we argue that this can make existing schemes unviable for many practical applications, (in particular, for BGPsec [Lep12] / Secure BGP [KLS00]) and present a new scheme based on trapdoor permutations like RSA that avoids this assumption. In fact, our scheme remains secure even if a signer does not *know* the public keys of the other signers.

1.1 Aggregate signatures from trapdoor permutations

Boneh, Gentry, Lynn, and Shacham [BGLS03] introduced the notion of aggregate signatures, in which individual signatures could be combined by *any third party* into a single constant-length aggregate. The [BGLS03] scheme is based on the bilinear Diffie-Hellman assumption in the random oracle model [BR93]. Subsequent schemes [LMRS04, Nev08] were designed for the more standard assumption of trapdoor permutations (*e.g.,* as RSA [RSA78]), but in a more restricted framework where third-party aggregation is not possible. Instead, the signers work *sequentially*; each signer receives the aggregate-so-far from the previous signer and adds its own signature.[1]

Lysyanskaya, Micali, Reyzin, and Shacham [LMRS04] constructed the first sequential aggregate signature scheme from trapdoor permutations, with a proof in the random oracle model.[2] However, their scheme has two drawbacks: the trapdoor permutation must be *certified* (when instantiating the trapdoor permutation with RSA, this means that each signer must either prove certain properties of the secret key or else use a long RSA verification exponent), and each signer needs to verify the aggregate-so-far before adding its own signature. Neven [Nev08] improved on [LMRS04] by removing the need for certified trapdoor permutations, but the need to verify before signing remained. Indeed, a signer who adds its own signature to an unverified aggregate in both [LMRS04] and [Nev08] (or, indeed, in any scheme that follows the same design paradigm) is exposed to a devastating attack: an adversary can issue a single malformed aggregate to the signer, and use the signature on that malformed message to generate a *valid* signature on a message that the signer never intended to sign (Appendix A).

The nonsequential scheme of [BGLS03] does not, of course, require verification before signing. The only known sequential aggregate scheme to not require verification before signing is the history-free construction of Fischlin, Lehmann, and Schröder [FLS11] (concurrent with our work), but it, like [BGLS03], requires bilinear Diffie-Hellman.

Thus, the advantages of basing the schemes on trapdoor permutations (particularly a more standard security assumption and fast verification using low-exponent RSA) are offset by the disadvantage of requiring verification before signing. We argue below that this disadvantage is serious.

[1] The need for the random oracle model was removed by Lu, Ostrovsky, Sahai, Shacham, and Waters [LOS$^+$06], who constructed sequential aggregate signatures from the bilinear Diffie-Hellman assumption; however, it is argued in [CHKM10] that this improvement in security comes at a considerable efficiency cost. See also [RS09, CSC09] for other proposals based on less common assumptions.

[2] Bellare, Namprempre, and Neven [BNN07] showed how the schemes of [BGLS03] and [LMRS04] can be improved through better proofs and slight modifications.

1.2 The need for lazy verification

In applications with a large number of possible signers, the need to verify before signing can introduce a significant bottleneck, because each signer must retrieve the public keys of the previous signers in order to run its signing algorithm. Worse yet, signers need to keep their large caches of public keys secure and current: if a public key is revoked and a new one is issued, the signer must first obtain the new key and verify its certificate before adding its own signature to the aggregate.

A key application: BGPsec. Sequential aggregate signatures are particularly well-suited for BGPsec [Lep12] (formerly known as the Secure Border Gateway Protocol (S-BGP) [KLS00]), a protocol being developed to improve the security of the global Internet routing system. (This application was mentioned in several works, including [BGLS03, LOS$^+$06, Nev08], and explored further in [ZSN05].) In BGPsec, autonomous systems (ASes) digitally sign routing announcements listing the ASes on the path to a particular destination. An announcement for a path that is n hops long will contain n digital signatures, added *in sequence* by each AS on the path.

Notice that the length of a BGPsec message *even without the signatures* increases at every hop, as each AS adds its name to the path, as well as extra information to the material in the routing message, such as its "subject key identifier" — a cryptographic fingerprint that is used to look up its public key in the PKI [Lep12]. Signatures add to this length. Long BGPsec messages may be problematic for two reasons: in transit, longer messages lead to packet fragmentation, which is a known security risk in IP networks (see [GH13] and references therein); and, at rest, routers (which are often memory constrained) need to store hundreds of thousands of BGPsec announcements in order to be able to forward them to the next hop whenever needed. Shorter signatures, and particularly aggregate signatures, can be used to mitigate this problem.

The BGPsec protocol is faced with two key performance challenges:

1. *Obtaining public keys.* BGPsec naturally requires routers to have access to a large number of public keys; indeed, a routing announcement can contain information from *any* of the 41,000 ASes in the Internet [COZ08] (this number is according to the dataset retrieved in 2012). Certificates for public keys are regularly rolled over to maintain freshness, and must be retrieved from a distributed PKI infrastructure [Hus12]. Caching more than 41,000 public keys is expensive for a memory-constrained device like a router (which often does not have a hard drive or other secondary storage [KR06]). Furthermore, whenever a router sees a BGPsec message containing a key that is not in its cache, it incurs non-trivial delay on certificate retrieval (from a distant device that hosts the PKI) and verification.

2. *Dealing with routing table "dumps".* When a link from a router to its neighboring router fails, the router receives a dump of the full routing table, often containing more than $300,000$ routes [CID], from it neighbors. Because routers are CPU- and memory-constrained devices, dealing with these huge routing table dumps incurs long delays (up to a few minutes, even with plain, insecure BGP [BHMT09]!). The delays are exacerbated if cryptographic signing and verifying is added to the process, and even more so when a router comes online for the first time (or after failure) and needs to also retrieve and authenticate public keys for all the ASes on the Internet.

To deal with these issues, the BGPsec protocol gives a router the option to perform *lazy verification*: that is, to immediately sign the routing announcement with its *own* public key, and to delay verification until a later time, *e.g.,* when (a) it has time to retrieve the public keys of the other signers, or (b) when the router itself is less overloaded and can devote resources to verification [DHS]. It is important to note that lazy verification by one router need not hurt others: if a router has not verified a given announcement, routers further in the chain can verify it for themselves.

While there is legitimate concern that permitting lazy verification may cause routers to temporarily adopt unverified paths, the alternative may be worse: forbidding lazy verification can lead to problems with global protocol convergence (agreement on routes in the global Internet), because of routers that delay their announcements significantly until they can verify signatures (*e.g.,* during routing table dumps, or while waiting to retrieve a missing certificate). Such delays create their own security issues, enabling easier denial of service attacks and traffic hijacking during the long latency window. Thus, even though BGPsec recommends that every router *eventually* verifies BGPsec messages, requiring that routers always verify *before* signing and re-announcing BGPsec messages is considered a nonstarter by the BGPsec working group [Sri12, Section 8.2.1]. Lazy verification is written into the BGPsec protocol specification as follows [Lep12, Section 7]:

> ...it is important to note that when a BGPSEC speaker signs an outgoing update message, it is not attesting to a belief that all signatures prior to it are valid.

Requirement: No public keys in the signing algorithm! Note that the primary obstacle here is *not* only verification time (which can perhaps be improved through batching and, anyway, can be considerably faster than signing time when using low-exponent RSA), but also the need to obtain public keys. Thus, lazy verification also requires that prior signers' public keys are *not* used in the signing algorithm (*e.g.,* hashed with the message as in [LMRS04, Nev08]).

Requirement: No security risk from signing unverified aggregates! As we already mentioned, a signer who adds its own signature to an unverified aggregate in the schemes of [LMRS04] and [Nev08] is exposed to a devastating attack (Appendix A). We already discussed how lazy verification may cause a signer to do so. Moreover, even without lazy verification, BGPsec may sometimes require a signer to add its own signature to an aggregate that is invalid. One such situation is when a router knowingly adopts a path that fails verification—for example, if it is the only path to a particular destination (the specification allows this [Lep12, Section 5]). It will then add its own signature to the invalid one, because a "BGPSEC router should sign and forward a signed update to upstream peers if it selected the update as the best path, regardless of whether the update passed or failed validation (at this router)" [Sri12, Section 8.2.1]. The need to sign a possibly invalid aggregate also arises in the case each message is signed by two different signature schemes (as will happen during transition times from one signature algorithm to another), and "one set of signatures verifies correctly and the other set of signatures fails to verify." In such a case the signer should still "add its signature to each of the [chains] using both the corresponding algorithm suite" [Lep12, Section 7]. Even if all BGPsec adopters avoid lazy verification and always verify before signing, these guidelines make it impossible to adopt an aggregate signature scheme that does not permit signing unverified aggregates, because of the possibility of attack. In other words, lazy verification is still needed for security even if no one uses it for efficiency!

Our goal. We note that lazy verification is permitted by the trivial solution of concatenating individual ordinary signatures, by aggregate signature schemes defined in [BGLS03], and by history-free aggregate signature schemes defined in [FLS11]. All of the above schemes do not require the current signer to know anything about the previous signers: neither their public keys nor the messages they signed.[3] Our goal is to obtain the same advantages, while relying on a more basic

[3]Identity-based aggregate signatures [YCK04, XZF05, CLW05, CLGW06, Her06, GR06, BGOY07, HLY09, SVSR10, BJ10] also remove the need for obtaining public keys and have been proposed for use in BGPsec. However, agreeing on the secret-key-issuing authority for the global Internet seems politically infeasible. Moreover, on a technical level, the proposals either require interaction among signers or are based on bilinear pairings. Interactive signatures would significantly complicate the protocol. And if we are willing to rely on bilinear pairings, [BGLS03] already gives us an excellent choice that allows for lazy verification.

security assumption than the bilinear Diffie-Hellman of [BGLS03, FLS11] and saving space as compared to the trivial solution.

1.3 Overview of our contributions

We present a sequential aggregate signature scheme that is secure even with lazy verification, based on any trapdoor permutation (such as RSA). Moreover, as in the nonsequential scheme of [BGLS03] and the history-free scheme of [FLS11], our signers do not need to know anything about each other—not even each other's public keys. To achieve this, we modify Neven's scheme [Nev08] by randomizing the H-hash function with a fresh random string per signer, which becomes a part of the signature, similarly to Coron's PFDH [Cor02] (Section 3). Our modification allows each signer to sign without verifying, and without even needing to know the public keys of all the signers that came before him, avoiding, in particular, the attack of Appendix A.

Although the ultimate goal in aggregate signatures is to produce schemes whose signature length is independent of the number of signers, signatures in our scheme grow slightly with the number of signers. However (as also pointed out by [Nev08]), while a constant-length aggregate signature is a theoretically interesting goal, what usually matters in practice is the *combined* length of signatures and messages, because that's what verifiers receive: signatures rarely live on their own, separately from the messages they sign. And the combined length of messages, if they are distinct, grows linearly with the number of signers, so the total growth of the amount of information received by the verifier is anyway linear. What matters, then, is *how fast* this linear growth is; below we derive parameters that show it to be much smaller than when ordinary trapdoor-permutation-based signatures are used as in the trivial solution.

We make the following contributions:

Generic randomized scheme. We present the basic version of our scheme, which requires each signer to append a *truly random* string to the aggregate (Section 3). Our scheme is as efficient for signing and verifying (per signer) as ordinary trapdoor-permutation based signatures, like the Full-Domain-Hash (FDH, [BR93, Section4]). We prove security (Section 4) in the random oracle model, based on the same assumption of trapdoor permutations (or claw-free permutations for a tighter security reduction) as in [Nev08]. Our security proof is more involved, because the reduction cannot know the public keys of other (adversarial) signers during the signature queries. We should note that our proof technique also shows that Neven's scheme need not hash other signer's public keys in the signing algorithm (however, Neven's scheme still fails under lazy verification).

Shortening the randomness. We show that the per-signer random string can be shorter if it is made input-dependent (Section 5), ensuring that a given signer never produces two different signatures on the same input. The idea of input-dependent randomness has been used before in signature schemes (e.g., [KW03, Section 4]); however, our application requires a new combinatorial argument to show security.

Instantiating with RSA. Appendix F shows how to instantiate our schemes with practical trapdoor permutations like RSA, which have slightly different domains for different signers.

Specification, implementation, benchmarking, and practical considerations. We develop a full, parameterized step-by-step specification of the truly-random and input-dependent-random versions of our signature when instantiated with RSA. We then implement our specification

Synchronized aggregate signatures (identity-based ones of [GR06] and regular ones of [AGH10]) also allow for lazy verification, but require a common nonce for all signers that, if repeated, breaks the security of the scheme. Implementing such a nonce in BGPsec presents its own challenges, because each signer has to ensure it never reuses a nonce, or else its secret key is at risk. The schemes are also pairing-based.

as a module in OpenSSL (Section 6); the specification and implementation are available from [BGR11]. We compare our implementation's performance to other potential solutions that allow for lazy verification; namely, [BGLS03], and the "trivial" solution of using n RSA or ECDSA signatures (the two algorithms currently proposed for use in implementations of BGPsec [DHS]). When evaluating signatures schemes for use with BGPsec, we consider compute time as well as signature length. Thus, we show that our signature is shorter than trivial RSA when there are $n > 1$ signers and shorter than trivial ECDSA when there are $n > 6$ signers. (While our signature is longer than the constant-length [BGLS03] signature, it benefits from relying on the better-understood security assumption of RSA.) Moreover, our scheme enjoys the same extremely fast verify times as RSA.

2 Preliminaries

Sequential aggregate signature security. The security definition for aggregate signatures (both original [BGLS03] and sequential [LMRS04]) is designed to capture the following intuition: each signer is individually secure against existential forgery following an adaptive chosen-message attack [GMR88] regardless of what all the other signers do. In fact, we will allow the adversary to give the attacked signer arbitrary—perhaps meaningless—aggregate-so-far signatures during the signature queries, thus making them adaptive "chosen-message-and-aggregate" queries. We also allow the adversary, which we call "the forger," to choose the public keys of all the other signers and to place the single signer who is under attack anywhere in the signature chain in the attempted forgery. This single attacked signer does not know any public keys other than its own and does not verify any aggregate-so-far given by the attacker.

Our definition is almost verbatim from [LMRS04], with one important difference needed to enable lazy verification: the public keys and messages of previous signers are not input to the signing algorithm. Therefore, each signer, by signing a message, is attesting only to that message, not to the prior signers' messages and public keys. At a technical level, this change implies that in security game the forger, in its query to ith signer, is required to supply only the aggregate-so-far signature allegedly produced by the first $i-1$ signers, but not the messages or public keys with respect to which this aggregate was allegedly produced. And, of course, to be considered successful, the forger must use *a new message*—in other words, it is not enough to change a public key or message of someone else in the chain before the attacked signer (because such public keys and messages may not even be well defined during the attack). This definition is exactly the one that is satisfied by the trivial solution of concatenating n individual signatures (and therefore suffices, in particular, for BGPsec).

Formally, a sequential aggregate signature scheme consists of three algorithms. Key generation is a randomized algorithm that (given a security parameter) outputs a public-private keypair (PK, SK). Aggregate signing Sign takes as input a private key SK, a message m_i to sign, and a sequential aggregate signature σ_{i-1} on messages $m_1, \ldots, m_{i-1}$ under respective public keys $PK_1, \ldots, PK_{i-1}$ (if $i = 1$, this signature σ_0 is taken to be the empty string, denoted by a special character ϵ; note that neither $m_1, \ldots, m_{i-1}$ nor $PK_1, \ldots, PK_{i-1}$ are given to Sign in our version of the definition). Sign outputs a sequential aggregate signature σ_i on all i messages $m_1, \ldots, m_i$. The aggregate verification algorithm is given a sequential aggregate signature σ_n, messages $m_1, \ldots, m_n$, and public keys $PK_1, \ldots, PK_n$, and verifies the validity of the signature on the given messages under the given keys. The correctness requirement is the natural one: that an aggregate signature produced by n successive invocations of Sign on the appropriate inputs should verify as correct.

The security requirement is formulated as follows. The adversary F against such a scheme is given a single public key and access to sequential aggregate signing oracle on the corresponding secret key. The advantage of F, $\mathsf{Adv\,AggSig}_F$, is defined to be its probability of success in the following game.

Setup. A key pair PK, SK is generated. The aggregate forger F is provided with PK, the challenge key.

Queries. F requests sequential aggregate signatures to be produced with SK on messages of its choice. For each query, it supplies an (alleged) sequential aggregate signature σ and an additional message m to be signed by the oracle under key SK. It can be adaptive, i.e., use the results of previous queries in order to decide on the current query.

Response. Finally, F outputs n distinct public keys $PK_1, \ldots, PK_n$ for some integer n of its choice. One of these keys must equal PK, the challenge key. Algorithm F also outputs messages $m_1, \ldots, m_n$, and a sequential aggregate signature σ.

The forger wins if the sequential aggregate signature σ is a valid sequential aggregate signature on messages $m_1, \ldots, m_n$ under keys $PK_1, \ldots, PK_n$, if $PK = PK_{i_*}$ for some $1 \leq i_* \leq n$, and if σ is nontrivial, i.e., F never issued a query on the message m_{i_*}. Note that i_* need not equal n: the forgery can be made in the middle of the sequence. The probability is over the coin tosses of the key-generation algorithm, the signing algorithm, and F.

In the random oracle model, F can also issue adaptive queries to the random oracle during its attack.

Fischlin, Lehmann, and Schröder [FLS11] propose a stronger security definition for their "history-free" signatures (building on history-free MACs of [EFG$^+$10]), which prevents certain reordering and recombining of signatures. Their definition thus has a security property that the trivial solution of concatenating n individual signatures does not have. Although this security property is not needed in many applications (for example, in BGPsec reordering and recombining of signatures is prevented simply by the protocol message structure, where each message must, for the purposes of functionality, include all the signed information contained in previous messages), our signature scheme in fact also prevents reordering and recombining that are of concern to [FLS11]: see Appendix G.

Cryptographic primitives. We will use pseudorandom function [GGM86] which we define here for the sake of completeness.

A pseudorandom function family (PRF) [GGM86] is one in which a randomly chosen function is indistinguishable from a truly random function by an observer of input-output behavior. We will consider only PRFs with variable input lengths and a fixed output length ℓ_r. The formal definition we need is as follows: if $\mathsf{PRF}_{\mathsf{seed}} : \{0,1\}^* \to \{0,1\}^{\ell_r}$ is a family of functions indexed by seed and D is an adversary that outputs a single bit (also known as *distinguisher*), consider the following two experiments. In the first, seed is chosen at random (not shown to D), and D gets to ask for outputs of $\mathsf{PRF}_{\mathsf{seed}}$ on inputs of its choice. In the second, a completely random function $f : \{0,1\}^* \to \{0,1\}^{\ell_r}$ is chosen at random, and D gets to ask for outputs of f on inputs of its choice. We will say the insecurity of PRF is the absolute value of the difference between the probabilities that D outputs 1 in the two experiments. We will denote by $\varepsilon_{\mathsf{PRF}}(q, t)$ the maximum insecurity of PRF against any D who asks at most q queries and runs in time t.

We assume the reader is familiar with the trapdoor permutations [DH76]. We will say that the generation algorithm for a trapdoor permutation outputs a description of a bijective function π (the public easy direction of the trapdoor permutation) and its inverse π^{-1} (the secret hard direction). Claw-free permutations [GMR88, Section 6.2] are trapdoor permutations with an additional feature:

the generation algorithm also outputs a description of a bijective function ρ on the same domain as π, such that, given π and ρ, it is hard to find a pair x, z with $\pi(x) = \rho(z)$ (such a pair is a called a "claw").

3 Our basic signature scheme

The intuition behind our construction is as follows. Recall the random-oracle-based Full Domain Hash signature scheme [BR93], in which a message m is hashed to same length as the domain of a trapdoor permutation π, and the signature x is computed as π^{-1} applied to the hash value.

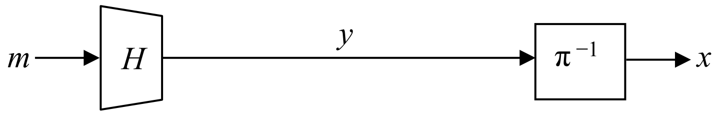

Figure 1: Full Domain Hash Signature Scheme of [BR93]

In order to aggregate such signatures, Lysyanskaya et al. [LMRS04] had the i^{th} signer XOR the previous signature x_{i-1} together with the hash value of the current message m_i and then apply the hard direction of the current signer's trapdoor permutation π_i^{-1} to get the signature x_i. The previous signature x_{i-1} could then be recovered during verification. Unfortunately, this approach had two drawbacks: it required (1) that the verification procedure check that each π_i is truly a bijection, and (2) that each signer verify the validity of the signature received from the previous signer.

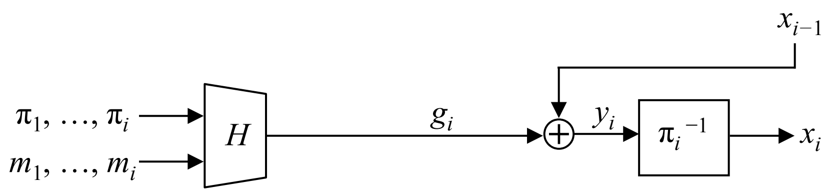

Figure 2: Aggregate Signature Scheme of [LMRS04]

Neven [Nev08] showed that the first drawback can be removed if the previous signature x_{i-1} is hashed together with current signer's message m_i. However, such hashing makes it impossible to recover x_{i-1} during verification of x_i. Neven then utilized the following approach from Bellare and Rogaway [BR96, Scheme PSS-R]: perform the hashing in two steps, first contracting to get a short value h_i, and then expanding to the domain of π_i. If the value h_i is available to the verifier, then the verifier can perform the second hashing step before even knowing x_{i-1}. Neven demonstrated that h_i values can be aggregated together by XORing. Thus, in Neven's scheme the signature output consists of two values: the output x_i of a trapdoor permutation and the short aggregated hash value h_i. (Neven has additional innovations to save more space and enable variable-size domains for the permutations π_i, which we omit here for simplicity of exposition.)

While the first drawback of the scheme of [LMRS04] is removed in Neven's scheme, the second one is still present: verifying before signing is necessary, because the transformation from (x_{i-1}, h_{i-1}) to (x_i, h_i) is deterministic, invertible, and can be performed by the adversary, except for the inversion of the trapdoor permutation performed at the last step. As we show in Appendix A,

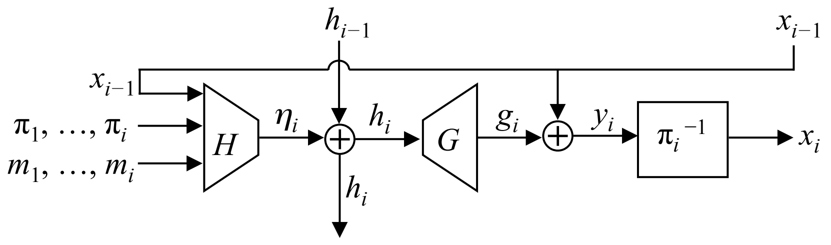

Figure 3: Aggregate Signature Scheme of [Nev08] (Simplified)

no scheme constructed in this manner can permit lazy verification while protecting against a chosen message attack. Thus, in our scheme, to enable lazy verification, we require each signer to add a random string r_i to the hash, and concatenate and append these strings to the signature. Because the adversary lacks a priori knowledge of these random strings, the chosen message attack becomes useless and we can prove that this is sufficient to enable lazy verification.

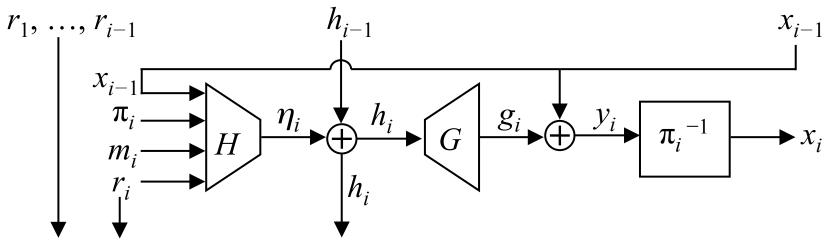

Figure 4: Our Aggregate Signature Scheme

Notation. We now describe the scheme precisely, using the following notation:

- Let m_i be the message signed by signer i.
- Let trapdoor permutation π_i be the public key of signer i and π_i^{-1} be the corresponding secret key. We assume all permutations operate on bit strings of length ℓ_π, *i.e.,* have domain and range $\{0,1\}^{\ell_\pi}$. (In Appendix F we remove the assumption that all permutations operate on the same domain. Section 6 uses this to instantiate π from the RSA assumption, where π_i is the easy direction, and π_i^{-1} is the hard direction of the RSA permutation.)
- Let H (*resp.* G) be a cryptographic hash function (modeled as a random oracle) that outputs ℓ_H-bit (*resp.* ℓ_π-bit) strings.
- Let ℓ_r be a parameter denoting the length of the randomness appended by each signer.
- Let the notation $\vec{a}_i$ denote a vector of values $(a_1, a_2, ..., a_i)$.
- Let $\oplus$ to denote bitwise exclusive-or. Exclusive-or is not the only operation that can be used; any efficiently computable group operation with efficient inverse can be used here.
- ϵ is a special character denoting the empty string; we assume $\epsilon \oplus x = x$ for any x.

Sign: The i^{th} Signer's algorithm

```
Require: $\pi_i, \pi_i^{-1}, m_i, x_{i-1}, h_{i-1}$ (where $x_{i-1}, h_{i-1} = \epsilon, \epsilon$ if $i = 1$).
1: Draw $r_i \xleftarrow{R} \{0,1\}^{\ell_r}$
2: $\eta_i \leftarrow H(\pi_i, m_i, r_i, x_{i-1})$
3: $h_i \leftarrow h_{i-1} \oplus \eta_i$
4: $g_i \leftarrow G(h_i)$
5: $y_i = g_i \oplus x_{i-1}$
6: $x_i \leftarrow \pi_i^{-1}(y_i)$
7: return $r_i, x_i, h_i$ {Note that $x_i$ and $h_i$ go to the next signer; all the $r_i$ values go to the verifier, but only the last signer's $x_i$ and $h_i$ do.}
```

Ver^{H,G}: The Verification Algorithm

```
Require: $\vec{\pi}_n, \vec{m}_n, \vec{r}_n, x_n, h_n$
1: for $i = n, n-1, ...., 2$ do
2:     $y_i \leftarrow \pi_i(x_i)$
3:     $g_i \leftarrow G(h_i)$
4:     $x_{i-1} \leftarrow g_i \oplus y_i$
5:     $\eta_i \leftarrow H(\pi_i, m_i, r_i, x_{i-1})$
6:     $h_{i-1} \leftarrow h_i \oplus \eta_i$
7: if $h_1 = H(\pi_1, r_1, m_1, \epsilon)$ and $\pi_1(x_1) = G(h_1)$ then
8:     return 1
9: else
10:    return 0
```

The i^{th} signer's signing algorithm has no dependency on the number of signers; it takes in *only* the i^{th} signers' own public key and message and the aggregated portion of the signature x_{i-1}, h_{i-1}. Moreover, the aggregated signature need not be verified before it is signed. For verification, only a single x_i and h_i—namely, the one from the last signer—is needed. However, every r_i, from the first signer to the last, is needed.

4 Security proof

We prove our scheme secure if G and H are modeled as random oracles and π is a trapdoor permutation. The proof is easier to understand if π is additionally claw-free (in particular, any homomorphic permutation, such as RSA, is claw-free if it is trapdoor). We therefore present the proof for the claw-free case. The more general case is addressed in Appendix E.

Our proof shows how a forger F on the aggregate signature scheme can be used to construct a reduction R that finds a claw in claw-free pair (π_*, ρ_*). R has F forge a signature for victim signer that uses permutation π_* by running $F(\pi_*)$, and then uses the resulting forgery $(\vec{\pi}_n, \vec{m}_n, \vec{r}_n, x_n, h_n)$ (where $\pi_* \in \pi_n$ and the value n is chosen by F) to find a claw in the claw-free pair. The structure of our reduction is similar to [Nev08]; however, while [Nev08] constructs a "sequential forger" from forger F and then constructs reduction R from the sequential forger, our reduction must proceed in one step (since the notion of a sequential forger is undefined if hash queries do not include previous signers public keys).

Simplifying assumptions about the forger F. The following simplifies our proof:

- We assume that the forger F forges the last signature in the signature chain; in other words, $\pi_n = \pi_*$ and m_n is a new message never queried by F to the signing oracle (whose public key is π_*). Indeed, any F can be easily modified to do so: if π_* and a new message $m_{n'}$ are present in $\vec{\pi}_n$ but at location $n' < n$, then we can run the verification algorithm loop for $n - n'$ iterations to obtain $x_{n'}, h_{n'}$ and output $(\vec{\pi}_{n'}, \vec{m}_{n'}, \vec{r}_{n'}, x_{n'}, h_{n'})$ as the new forgery, which will be valid if and only if the original forgery was valid. Note that we do not assume that π_* (or any other public key) is present in the signature chain only once.
- We assume that before forger F outputs its forgery and halts, it makes hash queries on all the hashes that will be computed during the verification of its forgery. Moreover, we assume that the forger does not output an invalid forgery; instead, it halts and outputs $\perp$. Indeed,

any F can be modified to do so; simply run the verification algorithm upon producing the forgery, and check that m_n is different from every message asked in a sign query.

F's queries. Note that when running F, the reduction R needs to be able to answer oracle queries by F, which are as follows:

- **H-Query.** F asks query $Q = (\pi, m, r, x)$ (where x may be ϵ) and expects to see $H(Q) = \eta$.
- **G-query.** F asks query h, and expects to see $g = G(h)$.
- **Sign Query.** F asks query (m, h, x) to be signed by π_*, and expects to see r, h', x' back, where r looks uniform, $h' = h \oplus H(\pi_*, m, r, x)$, and $\pi_*(x') = G(h') \oplus x$.

4.1 Description of the reduction R

4.1.1 Data structures used by R

HT and GT tables. The reduction R uses 'programmable random oracles', *i.e.,* it chooses answers for random oracle queries. R keeps track of queries whose answers have already been decided in two tables: HT for H and GT for G. We say $\mathsf{HT}(Q) = \eta$ if HT stores η as the answer to a query Q, and $\mathsf{HT}(Q) = \perp$ if HT has no answer for Q (similar for GT).

The HTree. The key challenge for the reduction is programming G, since G-queries are made on sums of H-query answers, rather than on individual H-query answers. Thus the reduction keeps an additional data structure, the HTree, that records responses to H-queries that may eventually be used as part of forger F's forgery. (HTree is inspired by the graph $\mathcal{G}$ in [Nev08, Lemma 5.3].)

The HTree is a tree of labeled nodes that stores a subset of the queries in HT. Each node in HTree (except the root) corresponds to an H-query that could potentially appear in the forger F's final forgery; the queries asked during verification of the forgery will appear on a path from one of the leaf nodes to the root (unless a very unlikely event occurs). The HTree has a designated *root node* that stores the value $h_0 = 0$. We consider the root to be at depth 0. A node N_i at depth $i > 0$ stores:

- a pointer to its parent node
- a query $Q_i = (\pi_i, m_i, r_i, x_{i-1})$ (where $x_{i-1} = \epsilon$ if and only if $i = 1$),
- the "hash-response" values $\eta_i = \mathsf{HT}(Q_i)$ and h_i, computed as the XOR of the values $\eta_1, \dots, \eta_i$ on the path from the root to the node N_i (equivalently, $h_{i-1} \oplus \eta_i$, where h_{i-1} is stored in the parent node),
- an auxiliary value y_i that is used to determine how future queries are added to the HTree, computed as $\mathsf{GT}(h_i) \oplus x_{i-1}$ (note that y_i is the value to which the signer would apply π_i^{-1}),
- if $\pi_i = \pi_*$, an auxiliary value z that may be used to find a claw in (π_*, ρ_*).

Every node at depth $i = 2$ or deeper satisfies the relation $\pi_{i-1}(x_{i-1}) = y_{i-1}$, where π_{i-1} and y_{i-1} are stored at the node's parent. New H-queries Q are added as nodes to the HTree if they can satisfy this relation; we say that such a query can be *tethered* to an existing node in the HTree. Intuitively, a query tethered to N_i becomes a child of N_i in the HTree:

Definition 4.1 (Tethered queries). *An H-query Q containing $x \neq \epsilon$ is* tethered *to node N_i in the HTree if N_i stores π_i, y_i such that $\pi_i(x) = y_i$. If $x = \epsilon$, then Q is tethered to the root of the HTree.*

The HTree's Lookup function (Algorithm 1) determines the HTree node to which query Q can be tethered. (Lemma B.3 argues that Lookup finds at most *one node* with high probability.) The HTree is populated via the Sim-H algorithm (Algorithm 4). The reduction R adds an H-query Q to the HTree if and only if it is tethered to some node in the HTree *at the time that forger F makes the H-query.* It is possible that some query Q is not tethered at the time it is made, but becomes tethered at at *later* time (after some new nodes are added to the HTree). However, Claim B.9 shows that this is highly unlikely.

4.1.2 Algorithms used by R

The reduction R uses the following algorithms (Algorithms 1–5).

G-queries. R answers these queries using a simple algorithm Sim-G (Algorithm 2). Sim-G returns $\mathsf{GT}(h)$ if it is already defined, or, if not, returns a fresh random value and records it in the GT.

Sign-queries The reduction R answers queries (m, h, x) to be signed by π_* using Sim-S (Algorithm 5). Since the reduction does not know the inverse of the challenge permutation π_*^{-1}, it 'fakes' a valid signature by carefully assigning certain entries in random oracle tables HT, GT, and ABORTS if these entries in HT, GT have been previously assigned. Later, we argue that Sim-S is unlikely to abort, since the entries added to HT, GT by Sim-S depend on a fresh random value r chosen as part of each signature query (Lemma B.7).

H-queries The reduction R answers these queries $Q = (\pi, m, r, x)$ using Sim-H (Algorithm 4). If there is an entry for Q in the HT, then Sim-H returns it. Otherwise, it assigns a fresh random value η as $\mathsf{HT}(Q)$. Next, Sim-H needs to prepare for the event that Q could lead to a forgery by the forger F, and thus needs to be stored in the HTree. To do this, Sim-H uses the Lookup function to check if Q can be tethered and thus should be added to the HTree. If Q can be tethered, Sim-H adds a new node to the HTree containing Q, its hash response η, and an auxiliary value y that is used by the Lookup function to tether future H-queries. In order to ensure that HTree is a tree (Lemma B.3), it is important to ensure that y is a fresh random value; Sim-H aborts if that's not the case. Finally, if Q contains the challenge permutation π_*, Sim-H adds a value z to the HTree node that FindClaw will use to derive a claw from a valid forgery output by the forger F. To prepare these values, Sim-H behaves almost as if it is 'faking' the answer to a sign-query, except that instead of using the usual challenge permutation π_* (as in Sim-S), it uses the challenge permutation ρ_* applied to z (so as to benefit from forger F's forgery, which would invert π_* on the output of $\rho_*(z)$, thus producing a claw). As in Sim-S, this involves carefully assigning certain entries in GT, and aborting if these entries are already assigned. (Claim B.6 shows that Sim-H is unlikely to abort.)

Finding a claw. Finally, forger F outputs a forgery $\vec{\pi}_n, \vec{m}_n, \vec{r}_n, x_n, h_n$, where $\pi_n = \pi_*$. Recall that our simplifying assumptions mean that the forgery is valid. The reduction R uses FindClaw (Algorithm 3) to find a claw from the forgery. Because we assumed all the queries for verifying the forgery have already been asked, the query $(\pi_*, m_n, r_n, x_{n-1})$ is in HT. Moreover, if the forgery is valid, then with high probability it is in the HTree as a child of the node storing $(\pi_{n-1}, m_{n-1}, r_{n-1}, x_{n-2})$, which is in turn a child of the node storing $(\pi_{n-2}, m_{n-2}, r_{n-2}, x_{n-3})$, *etc.* This holds because in a valid forgery, each H-query made during verification is tethered to the next one, and, by Claim B.9, all tethered queries are in the HTree with high probability. The value x_n (from the forgery) and value z_n (from HTree node of the query $Q = (\pi_*, m_n, r_n, x_{n-1})$) constitute a claw.

Algorithm 1 Lookup

```
Require: $x$
1: if $x = \epsilon$ then
2:     return Root node of HTree
3: else
4:     Nodelist = {all nodes $N$ in HTree containing $\pi, y$ such that $\pi(x) = y$}
5:     if Nodelist contains more than one node then
6:         ABORT
7:     else if Nodelist is empty then
8:         return $\perp$
9:     else
10:        return the single node in Nodelist
```

Algorithm 2 Sim-G: Answering a G-Query

```
Require: $h$
1: if $\mathsf{GT}(h) = \perp$ then
2:     Draw $g \overset{R}{\leftarrow} \{0,1\}^{\ell_\pi}$
3:     $\mathsf{GT}(h) \leftarrow g$
4: return $\mathsf{GT}(h)$
```

Algorithm 3 FindClaw

```
Require: $(\vec{\pi}_n, \vec{m}_n, \vec{r}_n, x_n, h_n)$ with $\pi_n = \pi_*$
1: $N_n \leftarrow \mathsf{Lookup}(x_n)$
2: if $N_n = \perp$ then
3:     ABORT
4: Retrieve $z_n$ the node $N_n$
5: return Claw $(x_n, z_n)$.
```

Algorithm 4 Sim-H: Answering an H-Query

```
Require: $Q = (\pi, m, r, x)$
1: if $\mathsf{HT}(Q) = \perp$ then
2:     Draw $\eta \overset{R}{\leftarrow} \{0,1\}^{\ell_H}$
3:     $\mathsf{HT}(Q) \leftarrow \eta$
4:     $N_{i-1} \leftarrow \mathsf{Lookup}(x)$
5:     if $N_{i-1} \neq \perp$ then
6:         Create new node $N_i$ with parent $N_{i-1}$
7:         Retrieve $h_{i-1}$ from parent $N_{i-1}$
8:         $h_i \leftarrow h_{i-1} \oplus \eta$
9:         if $\mathsf{GT}(h_i) \neq \perp$ then
10:            ABORT
11:        if $\pi \neq \pi_*$ then
12:            $g_i \leftarrow \mathsf{Sim\text{-}G}(h_i)$
13:            $y_i \leftarrow g_i \oplus x$
14:            Populate node $N_i$ with $Q, \eta, h_i, y_i$
15:        else
16:            Draw $z_i \overset{R}{\leftarrow} \{0,1\}^{\ell_\pi}$
17:            $y_i \leftarrow \rho_*(z_i)$
18:            Populate node $N_i$ with $Q, \eta, h_i, y_i, z_i$
19:            $\mathsf{GT}(h_i) \leftarrow y_i \oplus x$
20: return $\mathsf{HT}(Q)$
```

Algorithm 5 Sim-S: Answering a Sign-Query

```
Require: $(m, h, x)$
1: Draw $r \overset{R}{\leftarrow} \{0,1\}^{\ell_r}$
2: $Q \leftarrow (\pi_*, m, r, x)$
3: if $\mathsf{HT}(Q) \neq \perp$ then
4:     ABORT
5: else
6:     Draw $\eta \overset{R}{\leftarrow} \{0,1\}^{\ell_H}$
7:     $\mathsf{HT}(Q) \leftarrow \eta$
8: $h' \leftarrow \eta \oplus h$
9: Draw $x' \overset{R}{\leftarrow} \{0,1\}^{\ell_\pi}$
10: $y' \leftarrow \pi_*(x')$ .
11: if $\mathsf{GT}(h') = \perp$ then
12:     $\mathsf{GT}(h') \leftarrow y' \oplus x$
13: else
14:     ABORT
15: return $r, h', x'$.
```

4.2 Analysis of the reduction

Theorem 4.2. *If a forger F succeeds with probability ε, then the reduction R takes time at most $T(F) + q_S T(\textsf{Sim-S}) + q_H T(\textsf{Sim-H}) + q_G T(\textsf{Sim-G}) + T(\textsf{FindClaw})$ and finds a claw for (π_*, ρ_*) with probability at least*

$$\varepsilon - (q_S + q_H)(q_S + q_G + q_H)2^{-\ell_H} - q_S(q_S + q_H)2^{-\ell_r} - q_H^2 2^{-\ell_\pi}\,, \tag{1}$$

where $T(\cdot)$ denotes the running time of an algorithm, and q_H is the number of H-hash queries, q_G is the number of G-hash queries, and q_S is the number of sign queries made by the forger F.

We prove this theorem in Appendix B. The proof hinges on two key statements about the HTree. First (Claim B.5), the probability that $\mathsf{Lookup}(x)$ finds more than one HTree node is low (even though Lookup uses the functions π stored in the nodes of the HTree, which do not have to be permutations, because they are adversarially supplied and not certified like in [LMRS04]). Second (Claim B.9), an H-query that was not added to HTree is unlikely to become tethered at *some later time.* Both statements rely on the fact (proven in Claim B.4) that each time a query is placed on the HTree, its y value is random and independent of every other y value.

5 Shorter signatures via input-dependent randomness

To shorten our signature, we now show how to reduce ℓ_r (the length of the randomness appended by each signer). To do this, we replace the truly random r from our basic scheme with an r that is computed as a function of the inputs to the signer, and argue that it can be made shorter than the random r. Intuitively, we are able to maintain security with a shorter r because a given signer never produces two different signatures on the same input, thus limiting the information that an adversary can see and exploit. Of course, this input-dependent r need not be truly random; it suffices for a r to be a *pseudorandom* function of the input.

5.1 Modifying the scheme

We now compute r as a pseudorandom function (PRF) over the input (m_i, h_{i-1}, x_{i-1}) received by that signer i. Let $\mathsf{PRF}_{\mathsf{seed}} : \{0,1\}^* \to \{0,1\}^{\ell_r}$ be a PRF with seed seed and insecurity $\varepsilon_{\mathsf{PRF}}(q,t)$ against adversaries asking q queries and running in time t. Add a uniformly chosen seed to the secret key of the signer and replace line 1 of the signing algorithm with $r \leftarrow \mathsf{PRF}_{\mathsf{seed}}(m, h, x)$.

In the previous section, we found that ℓ_r must be long enough to tolerate a security loss of $q_S(q_H + q_S)2^{-\ell_r}$ (Theorem 4.2). As we show below, ℓ_r in the modified scheme can be shorter, since it needs only to allow for a security loss of approximately $(q_G + q_H + q_S + \ell_H q_S^2)2^{-\ell_r}$. This is an improvement if we assume that $q_H \approx q_G$ (since both H and G are hash functions) and $q_S \ll q_H$ (since in practice hash queries can be made offline, while signing queries need access to an actual signer).

5.2 Key insight for the security proof

Using the reduction of Section 4, we had to choose r long enough to make it unlikely that when a forger makes a sign query on $(\pi_*, m_i, x_{i-1}, h_{i-1})$, the algorithm $\mathsf{Sim\text{-}S}$ draws a random r_i that collides with a previously made H-query $Q_i = (\pi_*, m_i, r_i, x_{i-1})$. Indeed, if Q_i was answered by η_i and the forger chooses h_{i-1} so that h_i (which is computed as $h_{i-1} \oplus \eta_i$) has already been queried to G, then when r collides, the reduction would be prevented from programming the random oracle $G(h_i)$. Making r depend on the forger's input to the signer means that the forger gets only one chance (rather than q_S chances) to make this happen for a given Q_i, h_{i-1}, and h_i, because subsequent attempts by the forger will use the same r.

Proof Sketch. The full statement and proof of the security theorem for the modified scheme is in Appendix C. We modify the reduction R to not abort if HT already contains the relevant H-query. Specifically, we replace the ABORT on line 4 of $\mathsf{Sim\text{-}S}$ (Algorithm 5) with

$$\eta \leftarrow \mathsf{HT}(Q)\,.$$

(All the rest of R's algorithms are unchanged.) As such, we must now consider a new case where $\mathsf{Sim\text{-}S}$ aborts; namely, if $\mathsf{Sim\text{-}S}$ draws an r that defines a query $Q = (\pi_*, m, r, x)$ that is already in HT *and* the value $h' = \eta \oplus h$ is already stored in the GT (where recall that $\eta = \mathsf{HT}(Q)$ and h was given as part of the query to $\mathsf{Sim\text{-}S}$). Call such r *bad* for m, h, x. How likely is it that $\mathsf{Sim\text{-}S}$ draws a bad r?

Claim 5.1. $\Pr[\textit{Sim-S ever draws a bad } r] \leq (q_S + q_H)^2 2^{-\ell_H} + (q_G + q_H + q_S + (\ell_h + 2)q_S^2)2^{-\ell_r}$.

Proof. Consider a matrix ζ whose rows are indexed by queries (*i.e.,* h') in GT and whose columns are indexed by queries in HT that start with π_* (*i.e.,* , there is a column for each $Q = (\pi_*, m, r, x) \in \mathsf{HT}$). The entry in row h' and column Q is $h = h' \oplus \eta$, where $\eta = \mathsf{HT}(Q)$. $\mathsf{Sim\text{-}S}$ draws an r that is bad for m, h, x if and only if it (a) draws an r such that $Q = (\pi_*, m, r, x) \in \mathsf{HT}$, and (b) h exists in the Q^{th} column of ζ.

Thus, we will say that a column $Q = (\pi_*, m, r, x)$ of ζ is *bad* for (m, h, x) if at least one of the entries in that column is h (denote the set of such columns $BAD(m, h, x)$). The number of r values that are bad for a particular triple (m, h, x) is equal to the number of columns that are bad for that triple, and thus the probability that a bad r is chosen by $\mathsf{Sim\text{-}S}$ when responding to (m, h, x) is equal to $2^{-\ell_r} \cdot |BAD(m, h, x)|$. Now consider all the queries that have a given h. Note that the bad columns do not overlap for such queries (because each column is labeled with m and x). By the union bound, the probability that $\mathsf{Sim\text{-}S}$ draws a bad r during any signature query with h is at most $2^{-\ell_r} \cdot \sum_{m,x} |BAD(m, h, x)|$. Since the bad columns do not overlap, $\sum_{m,x} |BAD(m, h, x)|$ is bounded by the number of times h occurs in ζ. Thus, we can bound the overall probability that $\mathsf{Sim\text{-}S}$ ever draws a bad r by at most:

$$2^{-\ell_r} \sum_{i=1}^{q_S} \#\text{ of times the } i^{th} \text{ most frequent entry appears in } \zeta\,.$$

The claim follows from the answer to the following combinatorial problem. □

Combinatorial problem. Suppose β values $\eta_1, \ldots, \eta_\beta$ are chosen uniformly at random as ℓ_H-bit strings and given to an adversary, who then chooses α distinct values $h'_1, \ldots, h'_\alpha$. The $\alpha \times \beta$-matrix ζ is constructed by XORing the η and the h' values. A *collision* in ζ is a set of entries that are all equal. What is the total number of entries in the γ biggest collisions?

Theorem 5.2. *With probability at least* $1 - \beta^2 2^{\ell_H}$*, the total size of the* γ *biggest collisions in* ζ *is at most* $\alpha + (\ell_h + 2)\gamma^2$.

We can use Theorem 5.2 (proved in Appendix D) to bound the probability of choosing a bad r. α is the size of GT, which is at most $q_G + q_H + q_S$. β is the number of HT entries, which is at most $q_S + q_H$. γ is at most q_S. Then, the claim follows by observing that the probability that $\mathsf{Sim\text{-}S}$ ever aborts is at most (a) the probability that the event of Theorem 5.2 doesn't hold, which is at most

	2048-bit RSA	Our scheme	256-bit ECDSA	256-bit BGLS
Signature length (bits)	$2048n$	$2304 + 129n$	$512n$	257
Length for $n = 4.5$ (bits)	9216	2885	2304	257
Length for $n = 7$ (bits)	14336	3207	3584	257
Sign time (ms)	11.8	11.9	2.3	1.9
Verify time (ms)	$0.3n$	$0.3n$	$2.8n$	$\approx 18.9 + 6.6n$
Verify time for $n = 4.5$ (ms)	1.3	1.3	12.5	47.6
Verify time for $n = 7$ (ms)	2.1	2.1	19.4	64.8

Table 1: Benchmark results for n signers. Computed on a laptop with a Core i3 processor at 2.4GHz and 2GB RAM, running Ubuntu. The first three schemes were implemented using OpenSSL [ope] (with SHA-256 hashing and RSA public exponent of 65537); the BGLS scheme was implemented using MIRACL [Sco11] (with the curve BN-128 [BN05] and with precomputation on the curve generator but not on the public keys; further precomputation on the public keys seems to improve verification performance by up to 20% at the cost of additional storage). Results for specific values of n are not exactly in proportion due to rounding.

$\beta^2 2^{-\ell_H} = (q_S + q_H)^2 2^{-\ell_H}$, plus (b) the probability that, even though the event of Theorem 5.2 holds, a bad r is chosen, which is at most $(\alpha + (\ell_h + 2)\gamma^2)2^{-\ell_r} \leq (q_G + q_H + q_S + (\ell_h + 2)q_S^2)2^{-\ell_r}$. $\square$

6 Implementation and Evaluation

In Section F we present technical arguments for how our schemes can be instantiated with RSA. We implemented the input-dependent-r version as a module in OpenSSL [ope]. The full specification and the code are available from [BGR11].

Overview of our implementation. We instantiate the permutation π with 2048-bit RSA with public exponent 65537, hash H with SHA-256, full-domain hash G with the industry-standard Mask Generating Function (MGF) using SHA-256 [RSA02], and the pseudorandom function PRF with HMAC-SHA-256 [BCK96]. Instead of hashing the permutation π as-is inside the hash function H, we replace it with a short fingerprint of the RSA public key computed using SHA-256. Thus, we have parameters $\ell_\pi = 2048, \ell_h = 256$, and $\ell_r = 128$; the ℓ_r value is per signer, and each signer also adds one bit of information to deal with the problem that RSA gives each signer a slightly different domain (see Section F). Therefore, the length of the aggregate signature for n signers is $2048 + 256 + 129n$ bits long (see Table 1). We justify this choice of parameters as part of our specification, available from [BGR11].

Evaluation. We compare the implementation described in the previous paragraph to other signature schemes that allow for lazy verification. Table 1 contains data on our scheme as well as the "trivial" solution of using n RSA signatures, the solution of similarly using n ECDSA [Van92, IEE02] signatures (which are current contenders for adoption in BGPsec [Sri12, Section 4.1]), and the aggregate scheme of [BGLS03] (we do not compare against [FLS11], because it is a more complicated version of [BGLS03], so [BGLS03] performs better than [FLS11], anyway). In addition to providing formulas in terms of the number n of signers, we show results for specific values of $n = 4.5$ and $n = 7$. The value of 4.5 was chosen because it is roughly the average length of an AS path for a well-connected router on the Internet today (average length fluctuates with time and vantage point—see, e.g., here [Smi12]). We should note, however, that performance for higher than average values of n is particularly important: transition to BGPsec is expected to be particularly problematic for weaker routers, which are more likely to be located in the less well-connected portions of the Internet, and that experience longer than average paths. We therefore also show results for $n = 7$ (the smallest integer n for which our scheme beats ECDSA in signature length).

The table shows that the [BGLS03] scheme is a clear winner in terms of signature length and signing time, but has considerably slower verification[4]. It should be noted, however, that it is not being considered for the BGPsec standard at this stage [Sri12, Section 4.1]: schemes relying on bilinear Diffie-Hellman are not considered ready for worldwide deployment on the internet backbone by the BGPsec working group, because a consensus has not emerged on which curves provide the right tradeoff between security and efficiency (for example, there is not a NIST-approved set of curves such as the one contained in [NIS09, Appendix D] for non-pairing-based elliptic-curve cryptography). It is also important to note that the time required to compute group operations and bilinear pairings depends very heavily on the curve used; improvements for various curves are produced frequently, and there is no generally accepted set of curves or algorithms at this point. We believe that, assuming continued progress to speed-up pairings on specific curves and sufficient confidence in the security of bilinear Diffie-Hellman on these curves, the scheme of [BGLS03] (as improved by [BNN07]) should be considered for real applications.

As far as the remaining three schemes are concerned, we observe that ECDSA provides the shortest signatures when $n < 6$, while our scheme dominates the three for $n > 6$ (as we already mentioned, performance for higher than average n is particularly important.) We also observe that our scheme has computation time almost identical to simple RSA while having much shorter signatures (RSA signature length is listed as a particular concern in [Sri12, Section 4.1.2]). While ECSDA has the fastest signing time, the *verification* times for RSA and our scheme are an order of magnitude faster than those of ECDSA. Note that, for a router, the time required to sign does not depend on n, but the time required to verify grows linearly with n, so verification times are also of particular importance to weaker routers at the edge of the network.

Thus, if one is interested in a scheme based on the standard assumption of trapdoor permutations (albeit in the random oracle model), then our proposal fits the bill. Moreover, even if one is willing to accept security of ECDSA (which is not known to follow from any standard assumptions), our scheme may be preferable based on fast verify times and comparable-length signatures. Our scheme also has much faster verifying that pairing-based BGLS.

Acknowledgements

We thank Anna Lysyanskaya for help with the early stages of this work, Hovav Shacham and Craig Costello for helpful pointers and explanations, and the DHS S&T CSD Secure Routing project for many useful discussions that informed the design of our schemes. We thank anonymous referees for helping us improve our presentation and for pointing out related work. This work was supported by NSF Grants 1017907, 0546614, 0831281, 1012910, 1012798, and a gift from Cisco.

References

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 473–484. ACM, 2010.

[4]A more efficient pairing-based scheme of [WM08] with a constant total number of pairings was shown insecure by [SVS$^+$09].

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–32. Springer, 2003.

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 276–285. ACM, 2007.

[BGR11] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Implementation of sequential aggregate signatures with lazy verification, 2011. Available from `http://www.cs.bu.edu/fac/goldbe/papers/bgpsec-sigs.html`.

[BGR12] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 644–662. Springer, 2012.

[BHMT09] Zied Ben Houidi, Mickael Meulle, and Renata Teixeira. Understanding slow bgp routing table transfers. In *Proc. ACM SIGCOMM Internet measurement conference*, pages 350–355, New York, NY, USA, 2009. ACM.

[BJ10] Ali Bagherzandi and Stanislaw Jarecki. Identity-based aggregate and multi-signature schemes based on rsa. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 480–498. Springer, 2010.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 2007.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 12–16 May 1996. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[CHKM10] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Des. Codes Cryptography*, 55(2-3):141–167, 2010.

[CID] The CIDR report. `http://www.cidr-report.org`.

[CLGW06] Xiangguo Cheng, Jingmei Liu, Lifeng Guo, and Xinmei Wang. Identity-based multisignature and aggregate signature schemes from m-torsion groups. *Journal of Electronics (China)*, 23(4), July 2006.

[CLW05] Xiangguo Cheng, Jingmei Liu, and Xinmei Wang. Identity-based aggregate and verifiably encrypted signatures from bilinear pairing. In Osvaldo Gervasi, Marina L. Gavrilova, Vipin Kumar, Antonio Laganà, Heow Pueh Lee, Youngsong Mun, David Taniar, and Chih Jeng Kenneth Tan, editors, *ICCSA (4)*, volume 3483 of *Lecture Notes in Computer Science*, pages 1046–1054. Springer, 2005.

[Cor02] Jean-Sébastian Coron. Optimal security proofs for PSS and other signature schemes. In Lars Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 28 April–2 May 2002.

[COZ08] Ying-Ju Chi, Ricardo Oliveira, and Lixia Zhang. Cyclops: The Internet AS-level observatory. *ACM SIGCOMM CCR*, 2008.

[CSC09] Saikat Chakrabarti 0002, Santosh Chandrasekhar, Mukesh Singhal, and Kenneth L. Calvert. An efficient and scalable quasi-aggregate signature scheme based on lfsr sequences. *IEEE Trans. Parallel Distrib. Syst.*, 20(7):1059–1072, 2009.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DHS] Department of Homeland Security, Science and Technology Directorate, Cyber Security Division, Secure Protocols for Routing Infrastructure project. Personal Communication.

[DR02] Yevgeniy Dodis and Leonid Reyzin. On the power of claw-free permutations. In S. Cimato, C. Galdi, and G. Persiano, editors, *Third Conference on Security in Communication Networks SCN '02*, volume 2576 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2002.

[EFG$^+$10] Oliver Eikemeier, Marc Fischlin, Jens-Fabian Götzmann, Anja Lehmann, Dominique Schröder, Peter Schröder, and Daniel Wagner. History-free aggregate message authentication codes. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2010.

[FLS11] Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. Technical Report 2011/231, Cryptology ePrint archive, `http://eprint.iacr.org`, 2011.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GH13] Yossi Gilad and Amir Herzberg. Fragmentation considered vulnerable. *ACM Trans. Inf. Syst. Secur.*, 15(4):16, 2013.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.

[Her06] Javier Herranz. Deterministic identity-based signatures for partial aggregation. *Comput. J.*, 49(3):322–330, 2006.

[HLY09] Jung Yeon Hwang, Dong Hoon Lee, and Moti Yung. Universal forgery of the identity-based sequential aggregate signature scheme. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS*, pages 157–160. ACM, 2009.

[Hus12] G. Huston, editor. *The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)*. IETF RFC 6485, February 2012. Available from `http://tools.ietf.org/html/rfc6485`.

[IEE02] IEEE Std 1363-2000. IEEE standard specifications for public-key cryptography, 2002.

[KLS00] S Kent, C Lynn, and K Seo. Secure border gateway protocol (S-BGP). *J. Selected Areas in Communications*, 18(4):582–592, April 2000.

[KR06] Elliott Karpilovsky and Jennifer Rexford. Using forgetful routing to control bgp table size. In *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, pages 2:1–2:12, New York, NY, USA, 2006. ACM.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 155–164. ACM, 2003.

[Lep12] M. Lepinski, editor. *BGPSEC Protocol Specification.* IETF Network Working Group, Internet-Draft, July 2012. Available from `http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-04`.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2004.

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, 2006.

[Nev08] Gregory Neven. Efficient sequential aggregate signed data. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 52–69. Springer, 2008.

[NIS09] FIPS publication 186-3: Digital signature standard (DSS), June 2009. Available from `http://csrc.nist.gov/publications/PubsFIPS.html`.

[ope] OpenSSL toolkit. `http://openssl.org/`.

[RS09] Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo, editors, *ISA*, volume 5576 of *Lecture Notes in Computer Science*, pages 750–759. Springer, 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RSA02] *PKCS #1: RSA Encryption Standard. Version 2.1.* RSA Laboratories, June 2002. Available from `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[Sco11] Michael Scott. MIRACL library, 2011. `http://www.shamus.ie/`.

[Smi12] Philip Smith. BGP routing table analysis, 2012. `http://thyme.rand.apnic.net/`. See historical data—e.g., APNIC analysis summary for Sep. 7, 2012 at `http://thyme.apnic.net/ap-data/2012/09/07/0400/mail-global`.

[Sri12] K. Sriram, editor. *BGPSEC Design Choices and Summary of Supporting Discussions.* The Internet Engineering Task Force (IETF) Network Working Group, July 2012. `http://tools.ietf.org/html/draft-sriram-bgpsec-design-choices-02`.

[SVS$^+$09] S. Sharmila Deva Selvi, S. Sree Vivek, J. Shriram, S. Kalaivani, and C. Pandu Rangan. Security analysis of aggregate signature and batch verification signature schemes. Technical Report 2009/290, Cryptology ePrint archive, `http://eprint.iacr.org`, 2009.

[SVSR10] S. Sharmila Deva Selvi, S. Sree Vivek, J. Shriram, and C. Pandu Rangan. Identity based partial aggregate signature scheme without pairing. Technical Report 2010/461, Cryptology ePrint archive, `http://eprint.iacr.org`, 2010.

[Van92] Scott Vanstone. Responses to NIST's proposal. *Communications of the ACM*, 35:50–52, July 1992.

[WM08] Yiling Wen and Jianfeng Ma. An aggregate signature scheme with constant pairing operations. In *CSSE (3)*, pages 830–833. IEEE Computer Society, 2008.

[XZF05] Jing Xu, Zhenfeng Zhang, and Dengguo Feng. Id-based aggregate signatures from bilinear pairings. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 110–119. Springer, 2005.

[YCK04] HyoJin Yoon, Jung Hee Cheon, and Yongdae Kim. Batch verifications with id-based signatures. In Choonsik Park and Seongtaek Chee, editors, *ICISC*, volume 3506 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2004.

[ZBD05] Huafei Zhu, Feng Bao, and Robert H. Deng. Sequential aggregate signatures working over independent homomorphic trapdoor one-way permutation domains. In Sihan Qing, Wenbo Mao, Javier Lopez, and Guilin Wang, editors, *ICICS*, volume 3783 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2005.

[ZSN05] Meiyuan Zhao, Sean W. Smith, and David M. Nicol. Aggregated path authentication for efficient BGP security. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 128–138. ACM, 2005.

A Lazy verification and prior TDP-based proposals

[LMRS04] is a sequential aggregate signature that produces a constant length aggregate x_i. Using the notation of Section 3, the signature algorithm requires the i^{th} signer to compute the aggregate x_i from the aggregate-so-far x_{i-1} as follows:

$$x_i = \pi_i^{-1}(x_{i-1} \oplus H(\pi_1, ..., \pi_i, m_1, ..., m_i)) \tag{2}$$

The security of [LMRS04] relies on the fact that the i^{th} signer verifies x_{i-1} before producing x_i as above. We now show how an adversary can forge a valid signature on a message $\widehat{m}_i \neq m_i$ by adversarially malforming the aggregate-so-far.

Suppose the adversary knows that the i^{th} signer is signing the message m_i. In that case, playing the role of the first $i-1$ signers, he sets

$$\begin{aligned}\widehat{x}_{i-1} = x_{i-1} \oplus H(\pi_1, ..., \pi_i, m_1, ..., m_i)\\ \oplus H(\pi_1, \ldots, \pi_i, m_1, \ldots, \widehat{m}_i)\end{aligned}$$

where x_{i-1} is a valid signature on messages $m_1, \ldots, m_{i-1}$. If the i^{th} signer now produces a signature using the invalid aggregate-so-far $\widehat{x}_{i-1}$, then a little algebraic manipulation shows that the adversary now possesses a valid signature on messages $m_1, \ldots, m_{i-1}, \widehat{m}_i$, and thus the security of [LMRS04] does not hold under lazy verification. The analogue of this attack also works on [Nev08]. Here, the adversary malforms only the hash value h_{i-1} as:

$$\widehat{h}_{i-1} = h_{i-1} \oplus H(\pi_i, m_i, x_{i-1}) \oplus H(\pi_i, \widehat{m}_i, x_{i-1})$$

In fact, more generally, consider *any* scheme that, like [LMRS04] and [Nev08], operates as follows. The ith signer takes the aggregate-so-far (let's denote by it s_{i-1}), applies some transformation T to it that depends on the message m_i (let's denote it by $t_i = T_{m_i}(s_{i-1})$), and then applies π_i^{-1} to t_i (or a portion of it, as in [Nev08]) to get s_i. The verifier, for each i, takes s_i, applies the trapdoor permutation π_i to get $t_i = \pi_i(s_i)$, and inverts T_{m_i} to get $s_{i-1} = T_{m_i}^{-1}(t_i)$. Invertability of T, which is required for verification, is exactly the reason that such a scheme is insecure if s_{i-1} is not verified by signer i. The attack is the same as before: the adversary, playing the role of the first $i-1$ signers, computes a valid signature s_{i-1} on messages $m_1, \ldots, m_{i-1}$ and sets $\widehat{s}_{i-1} = T_{m_i}^{-1}(T_{\widehat{m}_i}(s_{i-1}))$, thus guaranteeing that $T_{m_i}(\widehat{s}_{i-1}) = T_{\widehat{m}_i}(s_{i-1})$. Now if signer i produces a signature s_i on message m_i using adversarially supplied $\widehat{s}_{i-1}$ as "signature" of signer $i-1$. then s_i is a valid signature on $m_1, \ldots, m_{i-1}, \widehat{m}_i)$, while signer i never intended to sign $\widehat{m}_i$.

This problem seems fundamental with the design paradigm: verification recovers every intermediate aggregate signature, so the function T_{m_i} needs to be invertible, but its invertability is what allows the attack. Any approach that tinkers with T_{m_i} will not result in a scheme that permits lazy verification. We thus need a different design paradigm—changing T_{m_i} to a randomized function is essentially what our scheme proposes.

B Proof of Theorem 4.2

Our proof proceeds in two steps. First, we argue that with high probability forger F will not detect that it is interacting with the reduction R rather than the real oracles. To do so, we define the *view* of forger F to be everything F sees during a run, including F's input (the public key), F's private coin flips, and the answers F receives to its H, G, and signature queries. The reduction, unlike the real execution, may abort before F outputs its forgery, in which cases we call the view of F "aborted." F's inability to detect that it is interacting with the reduction R follows from the following two lemmas.

Lemma B.1 (Correct Simulation). *For every view that is not aborted, the probability that F sees that view when interacting with the reduction is the same as the probability F sees that view in the real execution.*

Lemma B.2 (Abort Probability). *The reduction R aborts (either before or after the forger F outputs the forgery) with probability at most*

$$\Pr[ABORT] \leq (q_S + q_H)(q_S + q_G + q_H)2^{-\ell_H} + q_S(q_S + q_H)2^{-\ell_r} + q_H^2 2^{-\ell_\pi} \qquad (3)$$

Thus, if forger F outputs a forgery with probability ε when interacting with the real signer, then, by the union bound, the probability that it outputs a forgery and the reduction does not abort is at least $\varepsilon - \Pr[ABORT]$ (indeed, consider the union of two bad events: F doesn't output a forgery or the reduction aborts). But if the reduction does not abort, then it outputs a claw (x_n, z_n). By Lemma B.8, $\pi_*(x_n) = y_n$, where y_n is the value stored with the node N_n. And by construction of y_n on line 17 of Sim-H (Algorithm 4), $\rho_*(z_n) = y_n$. Hence, the reduction succeeds in finding a claw. It suffices to prove the two lemmas, which we do below.

B.1 Proof of Correct Simulation Lemma B.1

The public key given to forger F and F's private coin flips are the same in the simulation and in the real execution. Thus, it remains to show that the reduction R correctly simulates H-queries, G-queries, and Sign-queries. Indeed, consider a not aborted view of F obtained during interaction with reduction R.

- ***H*-queries.** For every H-query in the view, the answer was placed into HT by Sim-H (Algorithm 4) or Sim-S (Algorithm 5). Each produced the HT entry by drawing an independent uniformly random η, just like the real execution random oracle. The probability of that particular answer is $2^{-\ell_H}$ in both cases.

- ***G*-queries.** Consider a G-query in the view. Sim-G responded to that G query by returning a value from the GT. Values are assigned to the GT by Sim-G,Sim-H, and Sim-S, so it suffices to show that each of these algorithms assigns a fresh uniformly random value to the GT, which would guarantee that the particular value seen in the view was assigned with probability $2^{-\ell_\pi}$, just like the real execution random oracle. First, Sim-G assigns values g to GT by choosing them uniformly at random (Algorithm 2). Next, Sim-H assigns the value $g(h_i) = \rho_*(z_i) \oplus x$ to the GT, where z_i is a fresh random value, and ρ_* is a permutation (Algorithm 4). Since the domain of ρ_* is equal to the range of G, and ρ_* is a permutation, it follows that $\rho_*(z_i)$ is uniformly random in the range of G. It therefore follows that g is a uniform random value. Similarly, Sim-S assigns the value $g(h') = \pi_*(x') \oplus x$, for a fresh random value x' and a permutation π_*, so $g(h')$ is also a uniform random value.

- **Sign-queries.** Observe that, given the answers to the relevant H and G queries and the input (m, h, x), there is a unique correct pair h', x' for every r. The value r appears in the view as the result of a uniformly random choice in both the simulated execution and the real one. The answers to the relevant H and G queries are also results of uniformly random choices in both cases, by the arguments made for H- and G-queries. And h', x' are the unique correct values in both cases.

B.2 Proof of Abort Probability Lemma B.2

It suffices to compute the probability that Lookup, Sim-H, Sim-G, Sim-S, and FindClaw abort and to add them up by union bound. Sim-G never aborts. The following four lemmas address the remaining four algorithms, in order.

Lemma B.3. *The probability that Lookup ever aborts during the whole execution is at most*

$$\frac{q_H^2}{2} 2^{-\ell_\pi} \,. \tag{4}$$

Proof. Lookup(x) aborts when the HTree contains two nodes, N_1 and N_2, such that a Lookup collision occurs, *i.e.,*

$$\pi_1(x) = y_1 \text{ and } \pi_2(x) = y_2 \,. \tag{5}$$

Because the size of the HTree is bounded by q_H, it suffices to show that, for every pair of nodes N_1, N_2, the probability that there exists x such that Equation 5 holds is at most $2^{-\ell_\pi}$. We will do so by proving two claims.

Claim B.4. *When a node N_i storing π_i, y_i is added to the HTree, then y_i is chosen uniformly at random and independent of all prior choices made in the interaction between the forger F and the reduction R.*

Proof. Recall that nodes are added to the HTree by Sim-H. From Sim-H, if $\pi_i = \pi_*$, then we can think of y_i as fresh random value, since $y_i \leftarrow \rho_*(z_i)$, where z_i is a fresh random value and ρ_* is a permutation. If $\pi_i \neq \pi_*$, then y_i is chosen in a slightly more complex manner. Note from Sim-H that $y_i \leftarrow$ Sim-G$(h_i) \oplus x$. Sim-G returns a uniform random value for each new input, and so it follows that y_i will be a fresh random value *as long as GT is not defined on h_i*. But if GT is defined on h_i, then Sim-H will abort, so N_i will not be added to the tree, anyway. □

Lookup(x) is unlikely to find more than one node. Next, we need to show that the probability that there exists an x such that Equation 5 holds is small. To do this, we need to bound the probability that there are nodes N_1 and N_2 in HTree storing π_1, y_1 and π_2, y_2 such that there exists x with $\pi_1(x) = y_1$ and $\pi_2(x) = y_2$. Note that forger F can *adversarially-choose* π_1, π_2 stored in nodes N_1 and N_2 (since F issues a H-query that sets the π_i stored at each HTree node). Indeed, we have the following process: F chooses π_1 first, and then is given a independent random y_1, then chooses π_2 with knowledge of π_1, y_1, and finally is given independent random y_2 (Claim B.4). Thus, we *cannot* assume that π_1, π_2 are *permutations*; however, we may assume that they are *functions*:

Claim B.5. *For* any *two functions π_1, π_2 with domain $\{0,1\}^{\ell_\pi}$, and two uniformly random values y_1, y_2 in $\{0,1\}^{\ell_\pi}$, there exists x such $\pi_1(x) = y_1$ and $\pi_2(x) = y_2$ with probability at most $2^{-\ell_\pi}$.*

Proof. Define the set of preimages of y_1 under π_1 as $S_{y_1} = \{x \mid \pi_1(x) = (y_1)\}$. Suppose $|S_{y_1}| = \alpha$. Then there are at most α choices of y_2 that will result in the event that there exists x such $\pi_1(x) = y_1$ and $\pi_2(x) = y_2$, because each element $x \in S_{y_1}$ gives rise to at most one $y_2 = \pi_2(x)$. Because y_2 is chosen uniformly from a set of size 2^{ℓ_π}, the probability that x satisfying $\pi_1(x) = y_1$ and $\pi_2(x) = y_2$ exists is at most $\alpha 2^{-\ell_\pi}$. Thus, the desired probability is at most

$$\sum_\alpha \alpha 2^{-\ell_\pi} \Pr_{y_1}[|S_{y_1}| = \alpha] = 2^{-\ell_\pi} \sum_\alpha \alpha \cdot \frac{|\{y_1 \text{ s.t. } |S_{y_1}| = \alpha\}|}{2^{\ell_\pi}}.$$

Observing that $\sum_\alpha \alpha \cdot |\{y_1 \text{ s.t. } |S_{y_1}| = \alpha\}| = |\text{Domain}(\pi_1)| = 2^{\ell_\pi}$, we get the desired bound. □

Lemma B.3 follows by combining Claims B.4 and B.5. □

Lemma B.6. *A single invocation of Sim-H aborts on line 10 with probability at most* $(q_S + q_G + q_H)2^{-\ell_H}$.

Proof. From Algorithm 4, we see that Sim-H aborts only if GT already stores some value for h_i. First observe that there are at most $q_G + q_S + q_H$ queries stored in the GT. (There are q_G G-queries, and every signing and H-query query adds at most one additional entry to GT.) Next, observe that $h_i = h_{i-1} \oplus \eta$ where η is a fresh random value. Thus, the probability that h_i collides with a value already in GT is bounded by $(q_G + q_S + q_H)2^{-\ell_H}$. □

Lemma B.7. *A single invocation of Sim-S aborts with probability at most* $(q_H + q_S)2^{-\ell_r} + (q_S + q_G + q_H)2^{-\ell_H}$.

Proof. There are two cases in which Sim-S aborts.

Abort due to H-collision. Sim-S on input (m, h, x) will abort on line 4 if it draws a random value r such that $Q = (\pi_*, m, r, x)$ exists in the HT. The number of entries in HT cannot exceed $q_H + q_S$, because those are the only queries that add entries (at most one each) to HT. Because r_i is a fresh random value, the probability it collides with one of these entries is bounded by $(q_H + q_S)2^{-\ell_r}$.

Abort due to G-collision. Sim-S will abort on line 14 if GT already stores some value for h'. The same argument as in Lemma B.6 shows that this happens with probability at most $(q_S + q_G + q_H)2^{-\ell_H}$. □

Lemma B.8. *The probability of abort on line 3 of FindClaw is at most* $\frac{q_H^2}{2}2^{-\ell_\pi}$. *And if the abort does not happen, then* $\pi_*(x_n) = y_n$, *where* y_n *is stored with* N_n.

Proof. Suppose the forger F outputs a forgery $(\vec{\pi}_n, \vec{m}_n, \vec{r}_n, h_n, x_n)$ with $\pi_n = \pi_*$ that is valid relative to HT, GT, which means that FindClaw is invoked.

Consider running the verification algorithm$\mathsf{Ver}^{\mathsf{HT},\mathsf{GT}}$ with the forgery as input. The verification algorithm asks a sequence of H-queries $Q_n, ..., Q_1$, where $Q_1 = (\pi_1, r_1, m_1, \epsilon)$ and $Q_i = (\pi_i, m_i, r_i, x_{i-1})$ for every $i = 2...n$. We know that all these queries have been asked by forger F and are therefore in HT. Let $\eta_n, \dots, \eta_1$ be the answers to these queries. Note that these queries could not have been placed into HT by Sim-S, because m_n is different from every message queried to Sim-S with π_*. Thus, they were asked by the forger to Sim-H.

The verification algorithm also asks a sequence of G-queries $h_n, \dots, h_1$, where $h_{n-1} = h_n \oplus \eta_{n-1}$, $h_{n-2} = h_{n-1} \oplus \eta_{n-1}, \dots, h_1 = h_2 \oplus \eta_2$. Because the forgery is valid, $h_1 = \eta_1$, and therefore $h_i = \bigoplus_{j=1}^{i} \eta_j$. Note that all these G queries are in GT.

Note that Q_1 is tethered to the root of the HTree by Definition 4.1, and will therefore be placed in the HTree by Sim-H with the value $h_1 = \eta_1$. Because the forgery is valid, the x_1 value in Q_2

must satisfy $\pi_1(x_1) = y_1$, where $y_1 = G(h_1)$. Thus, Q_2 is tethered to Q_1. That does not necessarily mean that Q_2 itself is in the HTree. However, if it is, then it has the values $h_2 = h_1 \oplus \eta_2$ and $y_2 = G(h_2 \oplus x_1)$ stored in it. Thus, if Q_2 is in the HTree, then Q_3 is tethered to Q_2, because x_2 in Q_3 must satisfy $\pi_2(x_2) = y_2$, because that condition on Q_3 is necessary in order for the verification algorithm to query Q_2. Similarly, if Q_3 is in the HTree, then Q_4 must be tethered to it. By induction, either there exists $i > 1$ such that Q_i is tethered to Q_{i-1} but is not in the HTree, or all $Q_1, \ldots, Q_n$ are in the HTree. In the latter case, $\mathsf{Lookup}(x_n)$, if it does not abort, will return the node for the query Q_n (because Q_n was asked during verification, which happens only if $\pi_n(x_n) = y_n$), and thus FindClaw will not abort.

H-queries are unlikely to get tethered after they are asked. Thus, we have shown that, if FindClaw doesn't abort, then $\pi_n(x_n) = y_n$ (note that $\pi_n = \pi_*$), and that FindClaw will never abort unless there exists a query Q that was asked to Sim-H, is tethered to another query in HTree, but is not in the HTree. The following claim bounds the probability that this happens to a single H-query. To bound the probability that a tethered H-query exists outside the HTree, we add up over all q_H queries, to obtain $\frac{q_H^2}{2} 2^{-\ell_\pi}$ by the union bound.

Claim B.9. *If an H-query did not get added to HTree (equivalently, if it was untethered at the time it was asked to Sim-H), then the probability it will ever become tethered is at most $q'_H 2^{-\ell_\pi}$, where q'_H is the number of H queries made after it.*

Proof. Consider queries as they are added to HT in order. Suppose j_0-th query $Q = (\pi, m, r, x)$ was added as the result of a query to Sim-H and was untethered at the time it was asked, *i.e.,* the HTree was such that $\mathsf{Lookup}(x) = \perp$. Now suppose that Q first becomes tethered after some j_1-th query, $Q' = (\pi', m', r', x')$, is placed in HT. From the definition of a tethered query, Q' must have been added to the HTree. Thus, we must have $j_1 > j_0$, because we never remove nodes from HTree (*i.e.,* we cannot have $j_1 < j_0$) and Q itself is not added to the HTree (thus, $j_1 \neq j_0$). Since nodes are added to the HTree only when Sim-H is called on a new query, it follows that Q' was added to HTree after forger F asked a new H-query, so that the following collision occurs:

$$\pi'(x) = y' \,. \tag{6}$$

But, y' is a uniform random value that is independent of π' and x (Claim B.4), so the collision in equation (6) occurs with probability $2^{-|y|} = 2^{-\ell_\pi}$. This holds for each of the q'_H queries that could have been asked after j_0, and the claim follows by the union bound. □

This concludes the proof of Lemma B.8 □

Finally, Sim-H is called at most q_H times, Sim-S is called at most q_S times, FindClaw is called once and thus Lemma B.2 holds by a union bound.

C Analysis of the Scheme with Input-Dependent Randomness

We present only the parts of the proof that are different from Theorem 4.2.

Changes to the reduction R. We will assume that forger F never makes the same signature query twice, because it would get the same result, anyway (formally, we can always modify F not to ask the same signature query twice by keeping a table of previously requested signature queries). Reduction R uses the same algorithms as before, except that we replace the ABORT on line 4 of Sim-S (Algorithm 5) with $\eta \leftarrow \mathsf{HT}(Q)$.

Theorem C.1. *If a forger F succeeds in forging a signature for the modified scheme with probability ε, then the modified reduction R finds a claw for (π_*, ρ_*) in about the same running time as F with probability*

$$\varepsilon - 2(q_S + q_H)(q_S + q_G + q_H)2^{-\ell_H} - q_H^2 2^{-\ell_\pi} - (q_G + q_H + q_S + (\ell_h + 2)q_S^2)2^{-\ell_r} - \varepsilon_{\mathsf{PRF}}(q_S, t)\,,$$

where q_H is the number of H-hash queries, q_G is the number of G-hash queries, q_S is the number of sign queries made by the forger F, and t is the running time of the forger and the reduction combined.

Changes to Lemma B.1. Because Line 1 of Sim-S uses a truly random rather than a pseudorandom r, the probabilities of non-aborting views are no longer the same. However, by a standard reduction to the security of the PRF, the probability that forger F produces a forgery from a nonaborting view in the simulation must be at least $\varepsilon - \varepsilon_{\mathsf{PRF}}$.

Changes to Lemma B.2. This lemma changes only in Lemma B.7. The new version is:

Lemma C.2. *The probability that, during any of the q_S queries, Sim-S aborts is at most*

$$q_S(q_S + q_G + q_H)2^{-\ell_H} + (q_S + q_H)^2 2^{-\ell_H} + (q_G + q_H + q_S + (\ell_h + 2)q_S^2)2^{-\ell_r}\,.$$

Proof. When does the modified Sim-S abort? When $h' = \eta \oplus h$ is in GT. There are two cases. First, if η (and thus h') is a fresh random value, then the same argument used in Lemma B.7 holds, so abort probability is at most $(q_S + q_G + q_H)2^{-\ell_H}$.

However, we must now consider a new case where Sim-S aborts; namely, if η is *not* a fresh random value. η will not be a fresh random value if Sim-S is given a sign query (π_*, m, x, h) and draws an r that defines a query $Q = (\pi_*, m, r, x)$ that is (a) already in HT *and* (b) the value $h' = \eta \oplus h$ is already stored in the GT (where recall that $\eta = \mathsf{HT}(Q)$ and h was given as part of the query to Sim-S). Call such r *bad* for the sign-query's m, h, x. In Claim 5.1 we argued that Sim-S draws a bad r with probability at most $(q_G + q_H + q_S + (\ell_h + 2)q_S^2)2^{-\ell_r}$ □

D Combinatorial interlude: A proof of Theorem 5.2

Here we solve the combinatorial problem of Section 5. We start with a prelude problem:

A Prelude Problem. Suppose β values $\eta_1, \dots, \eta_\beta$ are chosen uniformly at random as ℓ_H-bit strings and the $\beta \times \beta$ matrix θ is computed as $\theta_{ij} = \eta_i \oplus \eta_j$. The diagonal of θ has all zero entries. Can we bound the size C_θ of the biggest nonzero collision within θ?

Lemma D.1. *With probability at least $1 - \beta^2 2^{-\ell_H}$, all the η_j values are distinct and $C_\theta \le 2\ell_h + 4$.*

Proof. Since we are not considering the 0 collision, we can remove the diagonal from our consideration and, in fact, focus only on the upper triangle of elements above the diagonal (the elements below the diagonal are equal to them, so we will get a nonzero collision of size $2k$ in θ if and only if we have k elements colliding in the upper triangle). Note that entries in a given row or given column are always distinct, unless $\eta_i = \eta_j$ for some $i \neq j$, which happens with probability no more than $\frac{\beta^2}{2}2^{-\ell_H}$. Consider the event that there is a collision of size k in the upper triangle or the η_j values are not distinct; let p_k be its probability. We can consider all subsets of k entries of the upper triangle in two parts: those subsets in which at least two elements share a row or a column (which, taken altogether, are covered by the case of nondistinct η_j values), and those in which all columns are distinct and all rows are distinct. Taking a union bound over all k-element subsets then gives us

$$p_k \leq \frac{\beta^2}{2} 2^{-\ell_H} + \sum_{\substack{0 \leq j_1 < \cdots < j_k \leq \beta; \\ \text{distinct } i_1 < j_1, \ldots, i_k < j_k}} \Pr[\eta_{i_1} \oplus \eta_{j_1} = \cdots = \eta_{i_k} \oplus \eta_{j_k}] \,. \tag{7}$$

The constraint $i_1 < j_1, \ldots, i_k < j_k$ comes from the fact that we are only considering the upper triangle. Because, for every a, the index j_a is greater than $j_1, \ldots, j_{a-1}$ and therefore also greater than $i_1, \ldots, i_a$, and the value η_{j_a} was chosen uniformly at random, we get that the value η_{j_a} is independent of $\eta_{i_1} \oplus \eta_{j_1} = \cdots = \eta_{i_{a-1}} \oplus \eta_{j_{a-1}}$, and of η_{i_a}, and therefore the a^{th} element $\eta_{i_a} \oplus \eta_{j_a}$ of the subset is independent of all the previous elements of the subset. This implies that (subject to the distinctness requirement on the i_as and j_as), $\Pr[\eta_{i_1} \oplus \eta_{j_1} = \cdots = \eta_{i_k} \oplus \eta_{j_k}] = 2^{-\ell_H(k-1)}$.

We have thus bounded the probability that a given k-element subset with distinct rows and distinct columns is a collision. That is, we bounded each addend of the sum in Equation 7. How many such subsets are there? There are β rows, β columns, and we are choosing k distinct rows and k distinct columns, so there are at most $\binom{2\beta}{2k}$ of them. Thus, the sum has at most $\binom{2\beta}{2k}$ addends. Substituting into the above formula, we get

$$\begin{aligned} p_k &\leq \frac{\beta^2}{2} 2^{-\ell_H} + \binom{2\beta}{2k} 2^{-\ell_H(k-1)} \\ &\leq \frac{\beta^2}{2^{\ell_H+1}} + \left(\frac{\beta e}{k}\right)^{2k} 2^{-\ell_H(k-1)} \\ &= \frac{\beta^2}{2^{\ell_H+1}} + \left(\frac{\beta^2 e^2}{k^2 2^{\ell_H}}\right)^{k-1} \frac{\beta^2 e^2}{k^2} \\ &= \frac{\beta^2}{2^{\ell_H+1}} \left(1 + \left(\frac{\beta^2 e^2}{k^2 2^{\ell_H}}\right)^{k-1} \frac{e^2 2^{\ell_H+1}}{k^2}\right) \end{aligned}$$

Observe that we can assume $\beta^2/2^{\ell_H} < 1$ (otherwise, the statement of the lemma is vacuous). So if $k \geq 4$, then $k^2 > 2e^2$ and thus $\frac{\beta^2 e^2}{k^2 2^{\ell_H}} < \frac{1}{2}$. So set $k = \ell_H + 2 \geq 4$. We get

$$p_k < \frac{\beta^2}{2^{\ell_H+1}} \left(1 + \left(\frac{1}{2}\right)^{\ell_H+1} \frac{e^2 2^{\ell_H+1}}{k^2}\right) < \frac{\beta^2}{2^{\ell_H+1}} \cdot 2 = \frac{\beta^2}{2^{\ell_H}} \,.$$

Thus we have that with probability $< \frac{\beta^2}{2^{\ell_H}}$, we have a collision in the upper triangle of size $\ell_H + 2$. The lemma follows because of the symmetry of the matrix. □

We are now ready to solve the combinatorial problem of Section 5:

Proof of Theorem 5.2. Assume the event of Lemma D.1 happens (it happens with probability $1 - \beta^2 2^{-\ell_H}$). Consider the largest collision in ζ; suppose its size is c_1 and its value is v_1. It has all distinct rows (because η_j values are distinct by the assumption that Lemma D.1 holds). Therefore, without loss of generality, we can assume that the collision occurs in rows $1, \ldots, c_1$ of ζ (this is just for convenience of notation). Call rows $1, \ldots, c_1$ the first *layer* of ζ. Consider the ith row of the first layer and the entry in that row that participates in the collision. That entry has value v_1, and therefore $h'_i = \eta_j \oplus v_1$ for some η_j. Thus, each of the values $h'_1, \ldots, h'_{c_1}$ is simply some η value shifted by v_1, and therefore the first layer of ζ corresponds to some c_1 distinct rows of the matrix θ from the prelude problem (distinct because the problem statement requires the h' values to be distinct), except with v_1 added to all values. Therefore, every collision that does *not* have value v_1 in the first layer of ζ is also a nonzero collision in θ; by the assumption that the event of Lemma D.1 holds, it has size at most C_θ.

Consider now the second largest collision in ζ, of size c_2 and value v_2. By the same argument as before, it has c_2 distinct rows. This time, some of these rows may be in the *first* layer of ζ; however, there are no than C_θ such rows, because the first layer of ζ has no collisions with value not equal to v_1 of size greater than C_θ. Let c'_2 be the remaining rows; we know $c_2 \le c'_2 + C_\theta$ and can assume without loss of generality that these rows are $c_1 + 1, \dots, c_1 + c'_2$ (again, this is just for convenience of notation). Call these rows the *second layer* of ζ. Suppose some other collisions occur in the second layer of ζ. Using the same argument we made for the first layer, it follows that these collisions correspond to nonzero collisions in θ (obtained by adding v_2 to all the values) and thus have size at most C_θ.

In general, if we consider the ith largest collision in ζ of size c_i, up to $C_\theta(i-1)$ of its rows can be from previous layers (*i.e.,* up to C_θ from each of the previous layers). Let c'_i be the remaining rows (call them the i^{th} layer); we have $c_i \le c'_i + C_\theta(i-1)$. No other collision in the i^{th} layer is of size more than C_θ.

Thus, the total size of γ collisions is at most $c_1+\dots+c_\gamma \le c_1+c'_2+\dots+c'_\gamma+C_\theta(1+2+\dots+\gamma-1)$. Because $c_1, c'_2, \dots, c'_\gamma$ refer to sizes of nonoverlapping layers, their sum is at most α. The theorem follows by observing that $1+2+\dots+\gamma-1 < \gamma^2/2$ and by substituting $C_\theta \le \ell_h + 2$ from Lemma D.1. $\square$

E Proving Security without the Claw-Free Assumption

Our security proofs so far assumed that (π, ρ) is a claw-free pair. They need a slight adjustment if π is only a trapdoor permutation and ρ does not exist (see [DR02] for a discussion of the differences between claw-free and trapdoor permutations when used with random-oracle-based signature schemes).

In such a case, the goal of the reduction is to invert π_* on a given input y. The reduction changes as follows: Line 17 of Sim-H, instead of picking y_i using ρ_*, picks y_i at random, except for one randomly chosen H-query, when y_i is set to equal y. When the forger produces a forgery, the value x_n in the forgery will satisfy $\pi(x_n) = y_i$ for some y_i chosen on Line 17 of Sim-H. This holds for the same reasons as in the claw-free case. With probability $1/q_H$, this y_i is equal to y, and so $\pi_*(x_n) = y$, and the reduction can output x_n as a successful inversion of π_* on y.

Thus, the reduction and its analysis remain essentially the same, except that the security is no longer tight: the probability of success of the reduction is reduced by a factor of $1/q_H$. Therefore, the statements of Theorems 4.2 and C.1 can be easily modified if the assumption changes from claw-free to trapdoor permutations: the probability formula simply needs to be reduced by a factor of q_H.

F Handling Permutations with Different Domains, such as RSA

Similarly to the schemes of [LMRS04] and [Nev08], our scheme extends to the case when each signer's permutation has its own domain, as long as no domain is much larger than the intersection of all the domains. For instance, if we instantiate our scheme with RSA using 2048-bit moduli, then each signer's permutation domain will be a subset of $\{0,1\}^{2048}$. The intersection of all the domains, however, will be at least the set of all 2048-bit strings that begin with 0, and thus no domain is more than twice the intersection of all the domains. (Following ideas of Zhu, Bao and Deng [ZBD05], the scheme can also be generalized to the case of domains of very different sizes, such as when different signers use RSA moduli of different lengths; we do not present this generalization here, because we expect all the moduli to be of the same standardized length in a typical deployment.)

Changes to the Scheme To explain how we modify the scheme, we need to fix some notation. We will assume that the domains of all permutations are subsets of $\{0,1\}^{\ell_\pi}$, and that the intersection of all the domains contains some set D closed under $\oplus$ (recall that the operation does not have to be exclusive-or—any group operation over D will do). Furthermore, we will assume that there is an efficient and efficiently invertible bijection SPLIT that takes an element X of $\{0,1\}^{\ell_\pi}$ and produces two values b, x, with $x \in D$ and $b \in \{0,1\}^{\ell_b}$ with ℓ_b close to $\ell_\pi - \log_2 |D|$. (For the case of RSA described above, $\ell_b = 1$. The function SPLIT sets $b = 1, x = X - 2^{2047}$ if $X \geq 2^{2047}$, and $b = 0, x = X$ otherwise.)

We will change G to output elements of D instead of $\{0,1\}^{\ell_\pi}$. We will change Step 6 of the signing algorithm as follows:

$$X_i \leftarrow \pi_i^{-1}(y_i); (b_i, x_i) \leftarrow \text{SPLIT}(X_i)$$

The signing algorithm will output b_i in addition to x_i. The entire vector $\vec{b}_n$ will be input to the verifying algorithm, which will be modified as follows: Step 2 will be replaced with

$$X_i \leftarrow \text{SPLIT}^{-1}(b_i, x_i); y_i \leftarrow \pi_i(X_i) .$$

Changes to the Reduction The security reduction needs to modified as follows. Recall that the reduction relies on the HTree, which is built up so that a child node is always tethered to a parent node. We will change the definition of "tethered" (Definition 4.1): an H-query Q containing $x \neq \epsilon$ will be *tethered* to some node N_i in the HTree if that node contains π_i, y_i such that there exists $b \in \{0,1\}^{\ell_b}$ for which $\pi_i(\text{SPLIT}^{-1}(b, x)) = y_i$.

Lookup (Algorithm 1) and FindClaw (Algorithm 3) will need to try all possible values of b to combine with the given x in order to find X to which π can be applied. Thus Step 4 of Lookup becomes

$$\begin{aligned}\text{Nodelist} = \{&\text{all nodes } N \text{ in HTree containing } \pi, y \\ &\qquad \text{such that } \exists b \in \{0,1\}^{\ell_b} \text{ such that } \pi(\text{SPLIT}^{-1}(b, x)) = y\} .\end{aligned}$$

Step 5 of FindClaw becomes

$$\begin{aligned}&\text{Find } b \in \{0,1\}^{\ell_b} \text{ such that } \pi_*(\text{SPLIT}^{-1}(b, x_n)) = \rho_*(z_n)\}; \\ &\qquad \text{return claw } (\text{SPLIT}^{-1}(b, x_n), z_n).\end{aligned}$$

Sim-H and Sim-S (Algorithms 4 and 5) need to search for y_i and y', respectively, that are in D. Thus, Steps 16 and 17 in Sim-H need to be repeated until $y_i \in D$ (also, z_i should be drawn from Domain(π_*) rather than $\{0,1\}^{\ell_\pi}$). Similarly, Steps 9 and 10 need to repeatedly draw $X' \overset{R}{\leftarrow}$ Domain(π_*) until $y' = \pi_*(X')$ is in D; the output of Sim-S should include $(b', x') = \text{SPLIT}(X')$.

Finally, Step 2 of Sim-G (Algorithm 2) should draw g from D rather than $\{0,1\}^{\ell_\pi}$.

Changes to the Analysis The above changes to the reduction will cause it to run 2^{ℓ_b} times slower (thus, twice as slow for the RSA example).

The analysis undergoes the following changes. Lemma B.1 remains true, but the proof is a bit more delicate: when arguing about the correct simulation of G queries, we need to rely on the fact that the new procedures in Sim-H and Sim-S still produce an output for G that is uniform in D, because y_i in $\mathsf{Sim\text{-}H}$ and y' in $\mathsf{Sim\text{-}S}$ are uniform in D, because they are produced by sampling a uniform distribution until an element of D is found. Claim B.4 also remains true, using the same argument.

Equation 5 and Claim B.5 change as follows.

Claim F.1. *Given two functions π_1, π_2 whose domains are subsets of $\{0,1\}^{\ell_\pi} = \mathrm{SPLIT}^{-1}(\{0,1\}^{\ell_b} \times D)$ and two uniformly chosen random values y_1, y_2 in D, the probability that there exists x, b_1, b_2 such that*

$$\pi_1(\mathrm{SPLIT}^{-1}(b_1, x)) = y_1 \text{ and } \pi_2(\mathrm{SPLIT}^{-1}(b_2, x)) = y_2 \tag{8}$$

holds is at most $2^{3\ell_b - \ell_\pi}$.

Proof. Define the set of preimages of y_1 as $S_{y_1} = \{(b_1, x) \text{ such that } \pi_1(\mathrm{SPLIT}^{-1}(b_1, x)) = y_1\}$. Suppose $|S_{y_1}| = \alpha$. Then there are at most $2^{\ell_b} \cdot \alpha$ choices of y_2 for which there exist x, b_1, b_2 satisfying Equation 8, because each triple in (x, b_1, b_2) with $b_2 \in \{0,1\}^{\ell}b$ and $(b_1, x) \in S_{y_1}$ gives rise to at most one $y_2 = \pi_2(\mathrm{SPLIT}^{-1}(b_2, x))$. Because y_2 is chosen uniformly from a set of size $|D|$, the probability that x, b_1, b_2 satisfying Equation 8 exist is at most $\frac{2^{\ell_b} \cdot \alpha}{|D|}$. Thus, the desired probability is at most

$$\sum_\alpha \frac{2^{\ell_b} \cdot \alpha}{|D|} \Pr_{y_1}[|S_{y_1}| = \alpha] = \frac{2^{\ell_b}}{|D|} \sum_\alpha \alpha \cdot \frac{|\{y_1 \text{ s.t. } |S_{y_1}| = \alpha\}|}{|D|}.$$

Observing that $\sum_\alpha \alpha \cdot |\{y_1 \text{ s.t. } |S_{y_1}| = \alpha\}| = \sum_{y_1} |S_{y_1}| = |\mathrm{Domain}(\pi_1)|$, and that $|\mathrm{Domain}(\pi_1)|/|D| \le 2^{\ell_b}$, we get that the probability is at most $2^{2\ell_b}/|D|$. Further observing that $|D| \ge 2^{\ell_\pi - \ell_b}$, we get the desired bound. □

This change results in the corresponding change in Lemma B.3: the $\frac{q_H^2}{2} 2^{-\ell_\pi}$ probability gets replaced by $\frac{q_H^2}{2} 2^{3\ell_b - \ell_\pi}$.

Finally, Claim B.9 needs modification. Equation 6 gets replaced with

$$\exists b \in \{0,1\}^{\ell_b} \text{ such that } \pi'(\mathrm{SPLIT}^{-1}(b, x)) = y',$$

which is satisfied with probability $2^{\ell_b - \ell_\pi}$. Thus, the probability in the statement of Claim B.9 changes to $q'_H 2^{\ell_b - \ell_\pi}$ and the probability that $\mathsf{FindClaw}$ aborts (bounded in Lemma B.8) changes to $\frac{q_H^2}{2} 2^{\ell_b - \ell_\pi}$

The above changes result in the the $q_H 2^{-\ell_\pi}$ term in the formulas of Lemma B.2 and Theorems 4.2 and C.1 being replaced with $q_H 2^{3\ell_b - \ell_\pi}$. Because ℓ_b is much smaller than ℓ_π, this change has no material impact on the security of the scheme.

G Our Scheme and History-Free Signatures

The definition of mezzo aggregation unforgeability satisfied by history-free signatures [FLS11] requires that the signature scheme, in addition to preventing forgery on new messages, should also prevent certain "reordering and recombining." To be precise, it means that the scheme should prevent the adversary from outputting an aggregate signature on messages $(m_1, \ldots, m_n)$ such that

- there are two consecutive uncorrupted signers $i, i+1$, and
- signer i was asked by the adversary to produce a signature on m_i and output σ_i in response, and

- signer $i + 1$ was asked by the adversary to produce a signature on m_{i+1}, but *not* with the value σ_i as the aggregate-so-far.

(the authors of [FLS11] formulate their requirements differently, but this formulation can be easily seen equivalent).

Such a strict requirement cannot possibly be satisfied by any scheme in which signer $i+1$ ignores some components of σ_i, because the adversary could simply modify those components of σ_i when querying signer $i + 1$. Thus, our scheme, in which σ_i contains r_i (which could be sent directly to the verifier and simply bypass signer $i + 1$) does not satisfy the definition of [FLS11] simply for syntactic reasons.

Nevertheless, our scheme satisfies the definition of [FLS11] in spirit: it ensures m_i was queried to signer i *before* m_{i+1} was queried to signer $i + 1$, and that the output of signer i was used as an input to signer $i + 1$. Thus, our scheme also makes the same "reordering and recombining" of signatures impossible. We outline how this can be shown by a small case analysis.

Suppose the adversary outputs a forgery; let i be the first location where the reordering condition is violated. Let x_i be the value computed as part of the verification process. If x_i was never queried to signer $i + 1$, then our security reduction still works for signer $i + 1$ (a forgery on a new x_i will work just as well as a forgery on a new m_{i+1}, since x_i is also input to H, so Lemma B.8, which is the only place where we use that m_{i+1} is new, still holds). Thus, x_i must have been queried to signer $i + 1$.

Suppose, then, that x_i was output in response to a sign query for signer i *after* it was queried to signer $i + 1$. The probability of such an event is very low: in responding to the signature query, signer i will pick a fresh r value, which will ultimately determine the x value output by the signer. (To bound this probability, observe that in order for the adversary to hit the correct x_i, it has to either get a collision on the input to G during a signature query, which is already accounted for in our reduction, or get G to output the correct ℓ_π-bit string on a fresh random input, which happens with probability $2^{-\ell_\pi}$.)

The only remaining case is that x_i was never output in response to a signature query for signer i. That means that when signer i was queried on m_i and x_{i-1}, either the r_i value output by signer i or the h_{i-1} value supplied with the query was different from the one in the forgery. If r_i is different in the forgery than in the signing query, our security reduction still goes through for signer i, since a forgery on a new r_i will work just as well as a new m_i in Lemma B.8. Finally, if r_i is the same and h_{i-1} is different in the forgery than in the signing query, we observe that the query to signer i must have been on an invalid h_{i-1}. We therefore modify $\mathsf{Sim\text{-}S}$ (Algorithm 5) as follows: if the hash query Q is tethered, but not to the input value h, then produce η using $\mathsf{Sim\text{-}H}$. Thus, the hash query that is involved in the forgery—namely, the query $(\pi_i, m_i, r_i, x_{i-1})$ will be answered using $\mathsf{Sim\text{-}H}$ and therefore Lemma B.8 still goes through. The modification results in a tiny increase in the abort probability because of the possibility of an additional GT collision during the signature query and a slightly larger GT.