Sequential Aggregate Signatures with Lazy Verification from Trapdoor Permutations

Extended Abstract

Kyle Brogle[1], Sharon Goldberg[2], and Leonid Reyzin[2]

[1] Stanford University Department of Computer Science
Stanford, CA 94305 USA
`broglek@stanford.edu`
Work done while at Boston University

[2] Boston University Department of Computer Science
Boston, MA 02215 USA
`{goldbe,reyzin}@cs.bu.edu`

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 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.

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.[3]

Lysyanskaya, Micali, Reyzin, and Shacham [LMRS04] constructed the first sequential aggregate signature scheme from trapdoor permutations, with a proof in the random oracle model.[4] 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

[3] 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.

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

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 (we describe the attack in the full version of the paper [BGR11b]).

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.

1.2 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 before it can even begin 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 the 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 the 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 like its “subject key identifier” — a cryptographic fingerprint that is used to lookup its public key in the PKI [Lep12].) 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 its 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. 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. [5] Our goal is to obtain the same advantages, while relying on a more basic security assumption than the bilinear Diffie-Hellman of [BGLS03,FLS11] and saving space as compared to the trivial solution.

[5] 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.

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.

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 on [LMRS04,Nev08].

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. In the full version of the paper [BGR11b] we show how to instantiate our schemes with practical trapdoor permutations like RSA, which have slightly different domains for different signers.

Detailed specification. We provide a full, parameterized step-by-step specification of the truly-random and input-dependent-random versions of our signature when instantiated with RSA (see the full version of the paper [BGR11b], where we also provide guidelines on choosing parameters such as bit lengths).

Implementation, benchmarking and practical considerations. We implement our specification as a module in OpenSSL (Section 6); the implementation is available from [BGR11a]. 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 formal definition, presented in the full version [BGR11b], 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).

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 [BGR11b].

Cryptographic primitives. We will use pseudorandom functions [GGM86]; the definition is omitted here because it is standard, but is presented in [BGR11b] for the sake of completeness. We will denote by $\varepsilon_{\mathsf{PRF}}(q, t)$ the maximum insecurity of PRF against any distinguisher who asks at most q queries and runs in time t.

We assume the reader is familiar with the trapdoor and claw-free permutations; we will denote by π the easy direction of the trapdoor permutation, by π^{-1} the hard direction, and by ρ the function such that it is hard to find a "claw" x, z with $\pi(x) = \rho(z)$.

3 Our basic signature scheme

The intuition behind our construction is as follows. Like [Nev08], we use a random-oracle-based signature with message recovery, similar to PSS-R [BR96], as a basic building block. Signatures with message recovery embed a portion of the message into the signature, so it can be recovered on verification and does not need to be sent explicitly. In our case, the signature outputs two values: the output x of a trapdoor permutation and an additional hash value h. The i^{th} signer receives (x_{i-1}, h_{i-1}) from the previous signer and wants to sign a message m_i. To enable aggregation, we view (x_{i-1}, m_i) together as a "message" to be signed with message recovery: we apply the signature with message recovery to this pair, so that x_{i-1} is embedded into the signature and does not have to be sent explicitly. The h portions of the signatures are exclusive-ored together for aggregation.

So far, what we described is a slightly simplified version of the scheme from [Nev08]. Note that verifying before signing is necessary in this scheme, 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 [BGR11b], no scheme constructed in this manner can permit lazy verification while protecting against a chosen message attack. Thus, to enable lazy verification, we require each signer to add a random string to the message, and concatenate and append these strings to the signature. Because the adversary lacks a priori knowledge about these random strings, the chosen message attack becomes useless and we can prove that this is sufficient to enable lazy verification.

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 the full version [BGR11b] 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 $\boldsymbol{a_i}$ 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 $r_i \overset{R}{\leftarrow} \{0,1\}^{\ell_r}$
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: $\boldsymbol{\pi_n}, \boldsymbol{m_n}, \boldsymbol{r_n}, x_n, h_n$
 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 homorphic 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 the full version [BGR11b]. 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_*, and then uses the resulting forgery to find the 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).

F's queries. We review what forger F expects to see on each one of its queries:

– **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 $\pi_*(x') = G(h') \oplus x$.
– **Forgery.** Finally, F outputs a forgery, $\sigma = \boldsymbol{\pi_n}, \boldsymbol{m_n}, \boldsymbol{r_n}, x_n, h_n$ where $\pi_n = \pi_*$. (Value n is chosen by F).

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 $\boldsymbol{\pi_n}$ 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 $\sigma' = \boldsymbol{\pi_{n'}}, \boldsymbol{m_{n'}}, \boldsymbol{r_{n'}}, x_{n'}, h_{n'}$ as the new forgery, which will be valid if an only if σ was valid. Note that we do <u>not</u> 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.

4.1 Description of the reduction R

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 σ; the queries asked during verification of σ 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 η_i and h_i (h_i is the XOR of the values $\eta_1, \ldots, \eta_i$ 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 $G(h_i) \oplus x_{i-1}$ (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 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 determines the HTree node to which query Q can be tethered. We can argue that Lookup finds at most *one node* with high probability.) The HTree is populated via the Sim-H algorithm. 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, we show that this is highly unlikely.

Algorithms used used by *R* The reduction R uses the following algorithms, which are formally specified in the full version [BGR11b].

G-queries. R answers these queries using a simple algorithm $\mathsf{Sim\text{-}G}$. $\mathsf{Sim\text{-}G}$ 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 $\mathsf{Sim\text{-}S}$. 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. We are able to argue that $\mathsf{Sim\text{-}S}$ is unlikely to abort, since the entries added to HT, GT by $\mathsf{Sim\text{-}S}$ depend on a fresh random value r chosen as part of each signature query.

H-queries The reduction R answers these queries $Q = (\pi, m, r, x)$ using $\mathsf{Sim\text{-}H}$. If there is an entry for Q in the HT, then $\mathsf{Sim\text{-}H}$ returns it. Otherwise, it assigns a fresh random value η as $\mathsf{HT}(Q)$. Next, $\mathsf{Sim\text{-}H}$ 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, $\mathsf{Sim\text{-}H}$ uses the Lookup function to check if Q can be tethered and thus should be added to the HTree. If Q can be tethered, $\mathsf{Sim\text{-}H}$ 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, it is important to ensure that y is a fresh random value; $\mathsf{Sim\text{-}H}$ aborts if that's not the case. Finally, if Q contains the challenge permutation π_*, $\mathsf{Sim\text{-}H}$ adds a value z to the HTree node that $\mathsf{FindClaw}$ will use to derive a claw from a valid forgery output by the forger F. To prepare these values, $\mathsf{Sim\text{-}H}$ 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 $\mathsf{Sim\text{-}S}$), 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 $\mathsf{Sim\text{-}S}$, this involves carefully assigning certain entries in GT, and aborting if these entries are already assigned. We are able to show that $\mathsf{Sim\text{-}H}$ is unlikely to abort.

Finding a claw. Finally, forger F outputs a forgery $\boldsymbol{\pi_n}, \boldsymbol{m_n}, \boldsymbol{r_n}, x_n, h_n$, where $\pi_n = \pi_*$. Recall that our simplifying assumptions mean that the forgery is valid. The reduction R uses $\mathsf{FindClaw}$ to find a claw from the forgery. Because we assumed all the queries for verifying σ 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 all tethered queries are in the HTree with high probability. The value x_n (from the forgery σ) and value z_n (from HTree node of the query $Q = (\pi_*, m_n, r_n, x_{n-1})$) constitute a claw.

4.2 Analysis of the reduction

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

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

where 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 full version of the paper [BGR11b]. The proof hinges on two key statements about the HTree. First, the probability that 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, an H-query that was not added to HTree is unlikely to become tethered at *some later time.* Both statements rely on the fact 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 1). 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 computes 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 attemps by the forger will use the same r.

We show in the full version [BGR11b] that the problem of proving this modified scheme secure hinges on 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 2. *With probability at least* $1 - \beta^2 2^{\ell_H}$*, the total size of the* γ *biggest collisions in* ζ *is at most* $\alpha + (\ell_h + 2)\gamma^2$.

The proof of this theorem, as well as the entire security analysis of the modified scheme, are found in [BGR11b]

6 Implementation and Evaluation

In the full version of the paper [BGR11b] we present details of instantiating our scheme with RSA (these include, in particular, dealing with the problem of slightly different domains for each signer's permutation). We implemented the input-dependent-r version as a module in OpenSSL [ope]. The code is available from [BGR11a].

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. 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 in [BGR11b].

Evaluation. We compare the implementation described in the previous paragraph to other signature schemes that allow for lazy verification. Table 1 contains

	2048-bit RSA	Our scheme	256-bit ECDSA	256-bit BGLS
Signature length (bits)	$2048n$	$2304 + 129n$	$512n$	257
Length for $n = 4.5$	9216	2885	2304	257
Length for $n = 7$	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 $n = 4.5$	1.3	1.3	12.5	47.6
Verify time for $n = 7$	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.

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 to 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 table shows that the [BGLS03] scheme is a clear winner in terms of signature length and signing time, but has considerably slower verification[6]. 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

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

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 *ACM Conference on Computer and Communications Security*, 2010.

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 *ACM Conference on Computer and Communications Security*, pages 276–285. ACM, 2007.

BGR11a. 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`.

BGR11b. Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification, 2011. Full version and implementation code, available from `http://www.cs.bu.edu/fac/goldbe/papers/bgpsec-sigs.html`.

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.

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 et al., 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.

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

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.

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. 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)*, 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.

ZSN05. Meiyuan Zhao, Sean W. Smith, and David M. Nicol. Aggregated path authentication for efficient BGP security. In *ACM Conference on Computer and Communications Security*, pages 128–138. ACM, 2005.