Information and Computation 239 (2014) 356–376

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Sequential aggregate signatures with lazy verification from trapdoor permutations [☆]

Kyle Brogle, Sharon Goldberg, Leonid Reyzin [*]

Boston University Department of Computer Science, 111 Cummington Mall, Boston, MA 02215, USA

ARTICLE INFO

Article history:
Received 22 August 2013
Received in revised form 9 June 2014
Available online 15 July 2014

Keywords:
Aggregate signatures
RSA
Lazy verification
BGP

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.

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., [37,39] relied on a seemingly

[☆] An extended abstract of this paper appeared in [6].
[*] Corresponding author.
E-mail address: reyzin@cs.bu.edu (L. Reyzin).

http://dx.doi.org/10.1016/j.ic.2014.07.001
0890-5401/© 2014 Elsevier Inc. All rights reserved.

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 [36]/Secure BGP [33]) 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 [3] introduced the notion of aggregate signatures, in which individual signatures could be combined by *any third party* into a single constant-length aggregate. The [3] scheme is based on the bilinear Diffie–Hellman assumption in the random oracle model [11]. Subsequent schemes [37,39] were designed for the more standard assumption of trapdoor permutations (e.g., as RSA [43]), 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 [37] 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 [39] improved on [37] 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 [37] and [39] (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 [3] 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 [24] (concurrent with our work), but it, like [3], 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 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 [36] (formerly known as the Secure Border Gateway Protocol (S-BGP) [33]), a protocol being developed to improve the security of the global Internet routing system. (This application was mentioned in several works, including [3,38,39], and explored further in [55].) 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 [36]. 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 [26] 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 [18] (this number is according to the dataset retrieved in 2012). Certificates for public keys are regularly rolled over to maintain freshness, and must be

[1] The need for the random oracle model was removed by Lu, Ostrovsky, Sahai, Shacham, and Waters [38], who constructed sequential aggregate signatures from the bilinear Diffie–Hellman assumption; however, it is argued in [13] that this improvement in security comes at a considerable efficiency cost. See also [42,19] for other proposals based on less common assumptions.

[2] Bellare, Namprempre, and Neven [10] showed how the schemes of [3] and [37] can be improved through better proofs and slight modifications.