Synchronized Aggregate Signatures: New Definitions, Constructions and Applications

Jae Hyun Ahn
Dept. of Computer Science
Johns Hopkins University
arjuna@cs.jhu.edu

Matthew Green
Dept. of Computer Science
Johns Hopkins University
mgreen@cs.jhu.edu

Susan Hohenberger
Dept. of Computer Science
Johns Hopkins University
susan@cs.jhu.edu

ABSTRACT

An aggregate signature scheme is a digital signature scheme where anyone given n signatures on n messages from n users can aggregate all these signatures into a single short signature. Unfortunately, no "fully non-interactive" aggregate signature schemes are known outside of the random oracle heuristic; that is, signers must pass messages between themselves, sequentially or otherwise, to generate the signature. Interaction is too costly for some interesting applications.

In this work, we consider the task of realizing aggregate signatures in the model of Gentry and Ramzan (PKC 2006) when all signers share a synchronized clock, but do not need to be aware of or interactive with one another. Each signer may issue at most one signature per time period and signatures aggregate only if they were created during the same time period. We call this *synchronized* aggregation.

We present a surprisingly efficient synchronized aggregate signature scheme secure under the Computational Diffie-Hellman assumption in the standard model. Our construction is based on the stateful signatures of Hohenberger and Waters (Eurocrypt 2009). Those signatures do not aggregate since each signature includes unique randomness for a chameleon hash and those random values do not compress. To overcome this challenge, we remove the chameleon hash from their scheme and find an alternative method for moving from weak to full security that enables aggregation. We conclude by discussing applications of this construction to sensor networks and software authentication.

Categories and Subject Descriptors: K.6.5 [**Security and Protection**]: Authentication.

General Terms: Security, Algorithms.

Keywords: aggregation, batch verification, standard model.

1. INTRODUCTION

Aggregate signatures, as introduced by Boneh, Gentry, Lynn and Shacham [11], are digital signatures where any party, given n signatures on n messages from n users can combine all of these signatures into a single short signature. This primitive is useful in many applications where storage or bandwidth is at a premium, and thus one wants to reduce the total cryptographic overhead.

While a number of aggregate signature schemes have been proposed [11, 24, 23, 16, 8, 6, 26, 3], all but one of these schemes is secure only in the random oracle model. The sole scheme in the standard model, by Lu, Ostrovsky, Sahai, Shacham and Waters [23], permits only *sequential* aggregation [23]. Sequentially-aggregate signatures [24] are a variant where signatures can only be aggregated by sequentially passing the (partially formed) aggregate from one signer to the next.

While sequential aggregation is useful for some applications, such as Secure BGP routing [20, 23], it is inappropriate for many important applications where signers cannot be conveniently arranged in sequence. This includes any situation where signers may operate independently of one another, e.g., when archiving signed email messages or compressing signatures on software applications. We highlight two such applications in Section 6.

Our Contributions.

Our main goal is to improve the state of the art for aggregate signatures and to consider how they can best be used to secure important communications in practice.

Re-visiting the Gentry-Ramzan model. Since aggregate signatures have proven difficult to build, the vast majority of research effort in this area has gone into building schemes that require that signers have some knowledge of and interaction with each other during the creation of the aggregate, either by sequential or broadcast messages.

In 2006, Gentry and Ramzan [16] proposed a solution to get around this dependence on interaction. In their model, signers do not need to be aware of or pass any messages between themselves, provided that they have a global strategy for choosing a unique value w that is used during signing, e.g., w can be the current time period. Only signatures with the same w value can be aggregated. They argue that this model is useful and realistic for some interesting applications. They provided the first construction, in the random oracle model.

In this work, we revisit their model, calling it *synchronized* aggregation, and provide additional formalizations. Signers can issue at most one signature per time period[1] and only

[1] As described in Section 5, we can allow the signer to issue

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

signatures issued during the same time period can be efficiently aggregated.

If the signers' clocks become out of sync in our model, then there is no security loss, but there is an efficiency loss, i.e., the out-of-sync messages cannot be aggregated. If a signer's clock is turned back in time causing them to issue two signatures during the same time period, then the unforgeability guarantee may no longer hold in our model (and indeed, it will *not* hold in this case for our constructions.) Although, for example, a sensor network under this sort of active attack is likely to be compromised in other ways and it is better that the adversary recover only the key for this single node than a symmetric key shared by many sensors.

For many applications, the clock need only be loosely synchronized. For example, see Liang et al. [22] which requires that each sensor have a clock, so that it can report on soil moisture conditions every 30 seconds.

A New Construction. We present the first synchronized aggregate signature scheme which is provably secure in the standard model. Prior synchronized or full schemes, due to Boneh, Gentry, Lynn and Shacham [11] and Gentry and Ramzan [16], only offered heuristic security arguments in the random oracle model. Similarly, prior to this work, only one aggregate scheme with a standard security proof existed, due to Lu, Ostrovsky, Sahai, Shacham and Waters [23], and it required sequential interaction among the signers. In this work, we offer the best of both worlds, by showing for the first time how to build a non-interactive scheme with a standard security proof.

Our construction is based on the Computational Diffie-Hellman (CDH) assumption in bilinear groups. The scheme is practical: an aggregate signature requires two group elements and one integer, and user public keys require one group element. To verify an aggregate of N signatures, requires only $k + 3$ pairings, where k is a security parameter (which could be five in practice), plus two full exponentiations and N small exponentiations (which could be 32 bits each in practice.)

Applications. We have several applications in mind for this technology, including aggregation of multicast messages (as in a sensor network) and reduction of storage requirements for signed executable code, as highlighted in Section 6. These applications each share the property that signers will produce messages independently of one another (making sequential aggregation or multi-signatures inapplicable). We show that, in this scheme, much of the work of signing a message can be performed *before* the message is known. This is important for sensors and other low-power devices.

Comparison with Prior Work.

Interestingly, there seems to be an inherent need for coordination among the signing parties or some other method for combing the randomness in the signatures to allow aggregation. This has typically been created through interaction by the signers (either broadcast or sequential) or artificially in the random oracle model. We review what is known in these models and then discuss new progress that does not require either interaction or random oracles, in the presence of a synchronized clock.

x signatures per time period at a cost of x elements in the public key.

Full Aggregation with Random Oracles. Boneh et al. [11] presented the first practical aggregate signatures (BGLS) under the CDH assumption in the random oracle model. (Technically, it required that all messages be distinct, although the authors [11] suggested a way to overcome this restriction, which was later fully explored by Bellare, Namprempre and Neven [6].) Unfortunately, to date, this is the only known full aggregate signature scheme and it is in the random oracle model.

Sequential Aggregation. In *sequential* aggregation, introduced by Lysyanskaya, Micali, Reyzin and Shacham [24], the partially formed aggregate signature is passed from signer i to signer $i + 1$, who then adds his information to the aggregate before passing it to the next signer. They gave a construction in the random oracle model based on families of certified trapdoor permutations.

Subsequently, Lu, Ostrovsky, Sahai, Shacham and Waters [23] provided the first sequential aggregate signatures secure in the standard model under the CDH assumption.

In 2007, Boldyreva, Gentry, O'Neill and Yum [8] presented an identity-based sequentially aggregate signature in the random oracle model, under an interactive complexity assumption, which was later shown to be false [19]. This highlights the importance of keeping the complexity assumptions simple, which is a goal of this work. Boldyreva et al. subsequently provided an alternate construction under a different interactive complexity assumption [9]. In 2008, Neven [26] proposed sequentially aggregate signed data as a new primitive that minimizes the total amount of transmitted data, rather than just the signature length, based on uncertified claw-free permutations in the random oracle model.

Interactive Aggregation. In the *interactive* model, as introduced by Bellare and Neven [7] for multisignatures, the signing process is an interactive protocol, where the signers communicate with each other to create the signature. In the recent scheme of Bagherzandi and Jarecki [3], to produce an aggregate signature, each signer must broadcast a message to all other signers. This requires that each signer be aware of and wait for all other signers before issuing a signature. On small devices, where transmissions are costly, this interaction could be prohibitively expensive and could eliminate the efficiency benefits of the aggregation. Their construction is secure under RSA in the random oracle model. Sequential aggregation can be considered a special case of interactive aggregation, where the signers only need to transmit messages to one another in a linear sequence.

Synchronized Aggregation. Ideally, one would like to eliminate the need for signer interaction. In 2006, Gentry and Ramzan [16] proposed an aggregate signature scheme where no interaction between signers was required, provided that all signers have a global strategy for choosing a value w that is used during signing (w can be the current time period.) Only signatures with the same w value can be aggregated. In this non-interactive model, Gentry and Ramzan were able to provide an identity-based scheme under CDH in the random oracle model, where the verification time is much more efficient than BGLS.

What is known in the standard model? To our knowledge, only two aggregation schemes are known that do not require random oracles. The first is the sequential scheme of Lu et al. [23]. The second is presented in this work. We

Scheme	Type	Assumption	Model	PP Size	PK Size	Agg Size	Agg Verify (in Pairings)
BGLS [11]	full	CDH	ROM	$O(1)$	1	1	$N+1$
LMRS [24]	seq	cert TDP	ROM	–	–	–	–
LOSSW [23]	seq	CDH	standard	$O(1)$	$O(k)$	2	2
LOSSW [23]	seq	CDH	ROM	$O(1)$	3	2	2
BGOY [9]	seq	IBSAS-CDH	ROM	$O(1)$	identity	3	4
Neven [26]	seq	uncert CFP	ROM	–	–	–	–
BJ [3]	int	RSA	ROM	–	–	–	–
GR [16]	sync	CDH	ROM	$O(1)$	identity	3	3
Our Work (Sec. 4)	sync	CDH	standard	$O(k)$	1	3	$k+3$
Our Work (App. A)	sync	CDH	ROM	O(1)	1	3	4

Figure 1: Summary of Full, Sequential and Synchronized Aggregate Signatures. TDP stands for trapdoor permutation and CFP stands for claw-free permutation. Let N be the number of individual signatures and k be a special security parameter (which could be five in practice). Identity-based schemes have a public-key size of "identity". Sizes for the public parameters (PP), public keys (PK) and aggregate signatures (Agg) count group elements and integers.

remove the need for interaction by working in the synchronized model. Their public keys require $O(k)$ elements for security parameter k, whereas ours require only *one* group element. Since public keys may form part of the transmission overhead and the goal of aggregation is to reduce this overhead, our scheme may offer a significant advantage.

Overview of the Construction.

Our design goals are three-fold. We want an efficient aggregate signature that is: (1) non-interactive, (2) secure under a standard assumption, such as CDH, and (3) not in the random oracle model. For basic signature schemes, there are currently only two "short" signature schemes that are fully secure in the standard model under the CDH assumption: the Waters signatures [29] and the more recent Hohenberger-Waters (HW) signatures [18]. The standard-model, sequentially-aggregate signatures of Lu et al. [23] are based on the Waters signatures, but it is not clear how to get around the sequential restriction (even if synchronized clocks are assumed.)

Building on HW Signatures. The more recent (stateful) HW signatures do not aggregate well at first blush. Recall that in that construction, the public key is of the form $(g, g^a, u, v, d, w, z, h)$ where these are random generators of a bilinear group $\mathbb{G}$ of prime order p and the secret key is a. The signer keeps a counter s. To sign a message M, she increments s, chooses two random values $r, t \in \mathbb{Z}_p$ and outputs the signature $\sigma = (\sigma_1, \sigma_2, r, s)$ where

$$\sigma_1 = (u^M v^r d)^a (w^{\lceil \lg(s) \rceil} z^s h)^t \text{ and } \sigma_2 = g^t.$$

Verification checks that

$$e(\sigma_1, g) = e(g^a, u^M v^r d) \cdot e(\sigma_2, w^{\lceil \lg(s) \rceil} z^s h).$$

The $u^M v^r$ value is a chameleon hash of M with randomness r. Aggregating these signatures requires the compression or coordination of the r values. It is not clear how to do this without breaking the security (since in the proof the simulator must be free to choose an r *after* seeing the message that the adversary asks him to sign.)

Attempt One. Our first observation is that, if we remove the randomized chameleon hash, then we can elegantly aggregate signatures with the same s value. Unfortunately, the HW security proof is fundamentally dependent on this chameleon hash. Absent this hash, the simulator must be able to guess not only the time period s^* that the adversary will use in her forgery (only in the case that s^* is "small", the proof handles "large" values of s^* another way), but also the message M which the adversary will query on during time period s^*. Since the message space is exponentially large, the simulator will fail to guess M with high probability.

Attempt Two. To solve this, we propose an alternative technique that reduces an exponential message space to a set of k polynomially-sized subspaces (each of size 2^ℓ, for some ℓ logarithmic in the security parameter). Our simulator need only guess one message from one of these subspaces. In practice, this requires that we divide our messages into k ℓ-bit chunks. The public parameters are $(g, u_0, u_1, \ldots, u_k, w, z, h)$, which all signers share, and public keys are $pk_i = g^{a_i}$. A signature on message $M = M_1 \ldots M_k$ during time period s is of the form $\sigma = (\sigma_1, \sigma_2, s)$ where

$$\sigma_1 = (u_0 \prod_{i=1}^{k} u_i^{M_i})^a (w^{\lceil \lg(s) \rceil} z^s h)^t \text{ and } \sigma_2 = g^t$$

and t is random. The signatures aggregate nicely (as we show in Section 4), but are they provably secure?

Attempt Three. A natural idea is to focus on one $u_i^{M_i}$ part, which now has a small enough message space to guess from, and treat it like an original HW signature (minus the chameleon hash.) Unfortunately, our aggregate security does not follow from the unforgeability of HW [18]. For $\ell = 1$, the adversary could ask for signature queries on the binary messages $001, 010, 100$ and then forge on the message 000. Thus, no chunk (i.e., bit) of the forgery message is "new"; meaning that no part of the aggregate forgery is an HW forgery. Fortunately, we make a key observation: the above scheme is not only existentially-unforgeable with respect to adaptive chosen *message* attacks, but also with respect to adaptive chosen *state-message pair* attacks, provided that each state is used only once. That is, it is hard to produce a signature on any "new" (s, M) pair, not just on a new message M.

Final Solution. Our new proof strategy takes advantage of this. We guess the time period s^* that the adversary

will use in her forgery (only in the case that s^* is "small"; we continue to handle "large" s^* values as before) and then guess (1) which message block $\beta \in [1, k]$ will differ between the message queried during s^* and the forgery message and (2) the value of this ℓ-bit message chunk. We succeed in our guess with non-negligible probability. Then, we can use our guess to contrive values for the public parameters and challenge public key which allow us to base our security on CDH, while maintaining our ability to aggregate signatures and handle those issues throughout the proof. We finally note that although aspects of this construction seem superficially related to the signatures of Waters [29] (which do not fully aggregate), our proof techniques are quite different.

2. DEFINITIONS OF SECURITY

In an aggregate signature scheme, anyone given n signatures on n messages from n users can aggregate all these signatures into a single short signature. This aggregate signature (together with the n public keys and n messages) can be publicly verified to convince anyone that user i authenticated message i for $i = 1$ to n. This is also true for synchronized aggregate signatures except that we assume all signers have a synchronized clock and the following restrictions apply:

1. A signer can issue at most one signature per time period and keeps state to ensure this.

2. Only signatures created during the same time period can be aggregated.

Gentry and Ramzan [16] were the first to consider this "synchronized" setting in the context of aggregate signatures. In their model, they assumed that signatures were issued using a special random tag w (which could not be re-used) and that only signatures with the same tag could be aggregated. They left open how signers coordinated their choice of w. Here, we use what seems to us the most natural coordination strategy– a synchronized clock.

Definition 2.1 (Synchronized Aggregation). *A synchronized aggregate signature scheme is a tuple of six algorithms* (Setup, KeyGen, Sign, Verify, Aggregate, AggVerify) *such that*

Setup(1^λ) *: the setup algorithm outputs public parameters* pp.[2]

KeyGen$(1^\lambda, \mathsf{pp})$ *: the key generation algorithm outputs a keypair* (pk, sk)*. Without loss of generality, we will assume that* pk *and* sk *contain* pp.

Sign(sk, M, s) *: the signing algorithm takes in a secret key* sk*, a message* M*, the current time period* s*, and produces a signature* σ*.*

Verify(pk, M, σ) *: the verification algorithm takes in a public key* pk*, a message* M*, and a purported signature* σ*, and returns 1 if the signature is valid and 0 otherwise.*

[2]In some schemes, pp may be empty. These parameters are included to capture many practical scenarios where multiple public keys are generated from the same algebraic group (e.g., [11, 16, 6]).

Aggregate$((pk_1, M_1, \sigma_1), \ldots, (pk_N, M_N, \sigma_N))$ *: On input a sequence of public keys* $(pk_1, \ldots, pk_N)$*, messages* $(M_1, \ldots, M_N)$*, and purported signatures* $(\sigma_1, \ldots, \sigma_N)$*, it outputs an aggregate signature* σ_{agg} *or error message* $\perp$*.*

AggVerify$((pk_1, \ldots, pk_N), (M_1, \ldots, M_N), \sigma_{agg})$ *: On input a sequence of public keys* $(pk_1, \ldots, pk_N)$ *and messages* $(M_1, \ldots, M_N)$*, and a purported aggregate signature* σ_{agg}*, the aggregate-verification algorithm outputs 1 if* σ_{agg} *is a valid aggregate signature and 0 otherwise.*

The correctness *property states that the* Verify *and* AggVerify *algorithms will always output 1 when run on correctly generated inputs.*

Unforgeability.

We recall the definition of Boneh et al. [11], which extends the standard security notion of existential unforgeability with respect to chosen-message attacks as formalized by Goldwasser, Micali and Rivest [17], to the case of aggregate signatures. It is defined using the following game between a challenger and an adversary $\mathcal{A}$.

Setup: The challenger runs $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters pp. It runs $\mathsf{KeyGen}(1^\lambda, \mathsf{pp})$ a total of N times to obtain the key pairs $(pk_1, sk_1), \ldots, (pk_N, sk_N)$. The adversary is sent $(pk_1, (pk_2, sk_2), \ldots, (pk_N, sk_N))$, which include pp.

Queries: Proceeding adaptively, for each time period 1 to q, the adversary can request a signature on a message of its choice under sk_1, provided that at most one query is made per time period. The challenger responds to a query for M_i at time period $s_i \in [1, q]$ as $\mathsf{Sign}(sk_1, M_i, s_i)$.

Output: Eventually, the adversary outputs a response of the form $((pk_1, \ldots, pk_N), (M'_1, \ldots, M'_N), \sigma)$. It wins the game if:

1. M'_1 is not any of $M_1, \ldots, M_q$; and
2. $\mathsf{AggVerify}((pk_1, \ldots, pk_N), (M'_1, \ldots, M'_N), \sigma) = 1$.

We define $\mathsf{SigAdv}_{\mathcal{A}}$ to be the probability that the adversary $\mathcal{A}$ wins in the above game, taken over the coin tosses made by $\mathcal{A}$ and the challenger.

Definition 2.2 (Unforgeability). *A forger* $\mathcal{A}$ (t, q, N, ϵ)*-breaks an* N*-user aggregate signature scheme if* $\mathcal{A}$ *runs in time at most* t*,* $\mathcal{A}$ *makes at most* q *signature queries and* $\mathsf{SigAdv}_{\mathcal{A}}$ *is at least* ϵ*. An aggregate signature is* (t, q, N, ϵ)*-existentially unforgeable under an adaptive chosen message attack if there is no forger that* (t, q, N, ϵ)*-breaks it.*

Discussion.

In the unforgeability definition, without loss of generality, we assume that the first public key in the challenge list is the challenge key pk_1. We also require that the non-challenge public keys be chosen honestly instead of adversarially. Alternatively, we can operate in the Knowledge of Secret Key (KOSK) model, where users register their keys with a CA and prove some necessary properties of the keys at that time [4]. In our construction and its proof (as in [23]), the adversary may choose her own public key (i.e., g^a), if she

can prove knowledge of the corresponding secret key (i.e., a). In our proof, we assume there exists a public, honest clock function $\mathsf{clock}()$, but do not give the simulator control over it. We also assume that the Setup algorithm is run by a trusted party or realized via secure multiparty computation. In the Section 4 construction, if a malicious party executes Setup and knows the discrete logarithms of the u_i values base g, then she can forge messages.

Finally, we note that one might want to consider a (seemingly) "stronger" definition, where we relax the nontriviality condition of the unforgeability game to allow the forgery message, M_1', to have been previously queried to the signing oracle provided that it was not done during the same time period used in the forgery. Observe that this "stronger" notion can be achieved by any scheme satisfying our unforgeability definition by simply having the signer incorporate the time period into each message.

3. ALGEBRAIC SETTING

Notation. For sets X, let $x \leftarrow X$ denote selecting an element $x \in X$ uniformly at random. For algorithms $\mathcal{A}$, let $a \leftarrow \mathcal{A}(y)$ denote that $\mathcal{A}$ output a when run on input y.

Bilinear Groups. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. A *bilinear map* is an efficient mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ which is both: (*bilinear*) for all $g \in \mathbb{G}, h \in \mathbb{G}_2$ and $a, b \leftarrow \mathbb{Z}_p$, $e(g^a, h^b) = e(g, h)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}_1$ and h generates $\mathbb{G}_2$, then $e(g, h) \neq 1$. This is called an *asymmetric* bilinear map. For simplicity of notation, we will present our constructions using *symmetric* bilinear maps, where we treat $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$, assuming efficient isomorphisms between them. All of our constructions can operate in the less restrictive asymmetric setting, which often allows smaller group sizes in practice. We make the following assumption in a bilinear group.

Our sole complexity assumption is the standard CDH [13].

ASSUMPTION 3.1 (COMPUTATIONAL DIFFIE-HELLMAN). *An algorithm $\mathcal{A}$ solves the CDH problem in $\mathbb{G}$ with advantage ϵ if*

$$\Pr[g \leftarrow \mathbb{G}; a, b, \leftarrow \mathbb{Z}_p; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}] \geq \epsilon.$$

We say that the (t, ϵ)-CDH assumption holds in $\mathbb{G}$ if no t-time algorithm has advantage at least ϵ in solving the CDH problem in $\mathbb{G}$.

4. A SYNCHRONIZED CONSTRUCTION IN THE STANDARD MODEL

Our aggregate scheme is based on the stateful, CDH signatures of Hohenberger and Waters [18].

Setup(1^λ).

The setup algorithm selects a bilinear group $\mathbb{G}$ of prime order $p > 2^\lambda$. Let $Z \in O(\lambda)$ be the number of bits in the message space. Let ℓ, k be two security parameters such that $\ell k = Z$. This will logically divide the Z-bit message space into k chunks of ℓ bits each. (As we discuss shortly, in practice one might set $\lambda = 80$, use a collision-resistant hash function to map arbitrarily-long strings into $Z = 160$ bits and then set $\ell = 32$ and $k = 5$.) It chooses random elements $g, u_0, \ldots, u_k, w, z, h \in \mathbb{G}$. It outputs the public parameters as

$$\mathsf{pp} = (\ell, k, p, \mathbb{G}, g, u_0, \ldots, u_k, w, z, h).$$

All parties have access to a function $\mathsf{clock}()$ that on no input, returns the current time period.[3]

KeyGen(1^λ, pp).

The key generation algorithm takes as input the parameters pp and selects a random $a \in \mathbb{Z}_p$. It outputs the public key as $\mathrm{PK} = (\mathsf{pp}, g^a)$ and the secret key as $\mathrm{SK} = (\mathsf{pp}, a)$. It also initializes s_{prev} to be zero.

Sign(SK, $M \in \{0,1\}^Z, s$).

The message space is Z bits; to sign arbitrarily-long messages one could first apply a collision-resistant hash function. We assume the signer is given the value $s = \mathsf{clock}()$ as input to the algorithm. It keeps as internal state s_{prev} denoting the last time period on which it issued a signature. If $s_{prev} = s$ or $s \geq 2^\lambda$, then it aborts. Otherwise, it records the current time period as $s_{prev} := s$. Let $M = M_1 M_2 \ldots M_k$, where each block M_i is ℓ bits. The signing algorithm selects a random $t \in \mathbb{Z}_p$ and then outputs a signature on M under key SK and time period s as:

$$\sigma_1 = (u_0 \prod_{i=1}^{k} u_i^{M_i})^a \cdot (w^{\lceil lg(s) \rceil} z^s h)^t, \quad \sigma_2 = g^t, \quad s.$$

Verify(PK, $M, \sigma = (\sigma_1, \sigma_2, s)$).

The verification algorithm first makes sure that $0 < s < 2^\lambda$. If this is false, then it rejects. It parses $M = M_1 M_2 \ldots M_k$ and verifies the signature by checking that

$$e(\sigma_1, g) = e(g^a, u_0 \prod_{i=1}^{k} u_i^{M_i}) \cdot e(\sigma_2, w^{\lceil \lg(s) \rceil} z^s h).$$

Aggregate($(pk_1, M_1, \sigma_1), \ldots, (pk_N, M_N, \sigma_N)$).

Parse σ_i as $(\sigma_{i,1}, \sigma_{i,2}, s_i)$. The aggregation algorithm verifies that s_1 is the third element of σ_i for $i = 2$ to N. If any check fails, it outputs $\perp$. Otherwise, it parses σ_i as $(\sigma_{i,1}, \sigma_{i,2}, s)$ and computes

$$\gamma_1 = \prod_{i=1}^{N} \sigma_{i,1} \quad , \quad \gamma_2 = \prod_{i=1}^{N} \sigma_{i,2}$$

The aggregate signature is output as (γ_1, γ_2, s).

AggVerify($(pk_1, \ldots, pk_N), (M_1, \ldots, M_N), \sigma$).

The verification algorithm parses $\sigma = (\gamma_1, \gamma_2, s)$ and checks that $0 < s < 2^\lambda$. If this is false, it rejects. Let $M_i = M_{i,1} M_{i,2} \ldots M_{i,k}$, where each division is ℓ bits. The algorithm extracts $g^{a_i} \in pk_i$, computes

$$V := e(\prod_{i=1}^{N} g^{a_i}, u_0) \cdot \prod_{j=1}^{k} e(\prod_{i=1}^{N} g^{a_i M_{i,j}}, u_j)$$

[3]The function $\mathsf{clock}()$ need not measure actual time. It can be replaced by any global strategy for choosing a unique integer s in the range $[1, T]$, where T is some fixed polynomial in the security parameter. This is the same as the Gentry-Ramzan aggregate signatures [16] except that they allow T to be exponentially large.

and verifies the signature by checking that

$$e(\gamma_1, g) \quad = \quad V \cdot e(\gamma_2, w^{\lceil \lg(s) \rceil} z^s h).$$

Efficiency Discussion. Our signatures require two elements in $\mathbb{G}$ plus a (small) integer, which may already be included in the payload (i.e.,"x is my sensor reading at time s".) Unlike the sequential-aggregate signatures of Lu et al. [23] which are based on Waters signatures, we are able to move the u_i values to the public parameters and thereby have public keys of only one element in $\mathbb{G}$ (whereas theirs require $O(\lambda)$ elements). Security degrades linearly in k but exponentially in ℓ, so the sizes of the message chunks cannot be too big. In practice, one could first apply a collision-resistant hash function to obtain a 160-bit message, then set $k = 5$ and $\ell = 32$, which breaks up the message into five 32-bit message chunks [25]. This would keep the size of the public parameters to a reasonable 10 elements in $\mathbb{G}$; it would also allow anyone to verify an aggregate of n signatures using only 8 pairings and no hashes into $\mathbb{G}$.

Using Asymmetric Groups. Our above construction can be set in an asymmetric bilinear group, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We will not require homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ (in either direction) nor will we require the ability to hash into either group. This may allow a wider class of curve choices [15]. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p, g_1 generate $\mathbb{G}_1$ and g_2 generate $\mathbb{G}_2$. Then to set the public parameters one would choose random $u'_0, \ldots, u'_k, w', z', h' \in \mathbb{Z}_p$ and output

$$\begin{aligned}
&\mathsf{pp} = (\ell, k, p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \\
&(\{u_{i,1} = g_1^{u'_i}\}_{i\in[0,k]}, w_1 = g_1^{w'}, z_1 = g_1^{z'}, h_1 = g_1^{h'}) \in \mathbb{G}_1^{k+4}, \\
&(\{u_{i,2} = g_2^{u'_i}\}_{i\in[0,k]}, w_2 = g_2^{w'}, z_2 = g_2^{z'}, h_2 = g_2^{h'}) \in \mathbb{G}_2^{k+4}).
\end{aligned}$$

The public key would be $pk = g_1^a \in \mathbb{G}_1$ with secret key $sk = a$. Both signature elements σ_1, σ_2 would be in $\mathbb{G}_1$. Instantiating our construction using the appropriate choice of pairing-friendly elliptic curves [14], we obtain very short signatures (approximately 320 bits plus a short integer for the time period for an 80 bit security level). The subsequent security proofs would follow in the same manner based on the hardness of the co-Computational Diffie-Hellman problem, defined as follows. An algorithm $\mathcal{A}$ solves the co-CDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ with advantage ϵ if

$$\begin{aligned}
\Pr[g_1 \leftarrow \mathbb{G}_1; g_2 \leftarrow \mathbb{G}_2; a, b, \leftarrow \mathbb{Z}_p; \\
z \leftarrow \mathcal{A}(g_1, g_1^a, g_1^b, g_2, g_2^a, g_2^b) : z = g_1^{ab}] \geq \epsilon.
\end{aligned}$$

4.1 Security Analysis

We refer the reader to Section 1 for intuition on our construction and its proof strategy.

THEOREM 4.1. *Suppose the (t', ϵ')-CDH assumption holds in group $\mathbb{G}$ of prime order p. Then the aggregate signature scheme above is (t, q, N, ϵ)-secure against existential forgery under an adaptive chosen message attack provided that*

$$\epsilon \geq 2\epsilon' \cdot \max(\lambda, 2^{\ell+1} \cdot q \cdot k) \quad , \quad t \leq t' - \Theta(T(N + q + k))$$

where q is the number of signing queries, $Z = \ell \cdot k \in O(\lambda)$ is the message length, $2^\lambda < p$, and T is the maximum time for an exponentiation in $\mathbb{G}$ or $\mathbb{G}_T$.

PROOF. As in [18], we consider two types of adversaries, using either to solve CDH.

Type I The adversary forges for a message with period s greater than $2^{\lceil \lg(q) \rceil}$.

Type II The adversary forges for a message with period s less than or equal to $2^{\lceil \lg(q) \rceil}$.

Suppose that an adversary $\mathcal{A}$ breaks this signature with total probability ϵ, probability ϵ_1 when acting as a type I forger, and probability ϵ_2 when acting as a type II forger. Let P_1 (resp., P_2) be the probability that $\mathcal{A}$ chooses to be a type I (resp., type II) forger. Then $\epsilon = P_1 \cdot \epsilon_1 + P_2 \cdot \epsilon_2$.

In Lemma 4.2, we show that if $\mathcal{A}$ breaks this scheme as a type I forger with probability ϵ_1, then the simulator can break CDH with probability ϵ_1/λ. In Lemma 4.3, we show that if $\mathcal{A}$ breaks this scheme as a type II forger with probability ϵ_2, then the simulator can break CDH with probability $\epsilon_2/(2^{\ell+1} \cdot q \cdot k)$. The simulator chooses her setup parameters differently depending on which type of adversary she is interacting with. At the start of this game, let the simulator randomly choose which type of adversary she will interact with. She will guess correctly with probability exactly one half and abort otherwise. Both type I and type II simulations will be indistinguishable from the view of the adversary. Thus, the probability of solving the CDH problem is

$$\begin{aligned}
\epsilon' \quad &\geq \quad \frac{1}{2}\left(P_1 \cdot \epsilon_1 \cdot \frac{1}{\lambda} + P_2 \cdot \epsilon_2 \cdot \frac{1}{2^{\ell+1} \cdot q \cdot k}\right) \\
&\geq \quad \frac{P_1 \cdot \epsilon_1 + P_2 \cdot \epsilon_2}{2 \cdot \max(\lambda, 2^{\ell+1} \cdot q \cdot k)} \\
&= \quad \frac{\epsilon}{2 \cdot \max(\lambda, 2^{\ell+1} \cdot q \cdot k)}.
\end{aligned}$$

□

4.1.1 Type I Adversary

LEMMA 4.2. *Suppose the (t', ϵ')-CDH assumption holds in $\mathbb{G}$ of prime order p. Then the aggregate signature scheme above is (t, q, N, ϵ)-secure against type I existential forgery under an adaptive chosen message attack provided that*

$$\epsilon \geq \epsilon' \cdot \lambda \quad , \quad t \leq t' - \Theta(T(N + q + k))$$

where q is the number of signing queries, $Z = \ell \cdot k \in O(\lambda)$ is the message length, $2^\lambda < p$, and T is the maximum time for an exponentiation in $\mathbb{G}$ or $\mathbb{G}_T$.

PROOF. Given a CDH challenge (g, g^a, g^b), proceed as:

Setup.

The simulator begins by guessing a value α^* in the range 1 to λ. This represents a guess that the adversary will forge on some period s such that $\alpha^* = \lceil \lg(s) \rceil$. For type I adversaries, recall that if the adversary forges using a time period that maps to α^*, it will not ask enough signing queries to see an original signature with a time period that maps to α^*. The verification algorithm rejects on all indexes greater than or equal to 2^λ.

Next, choose random $y_1, \ldots, y_k \in \mathbb{Z}_p$ and set $u_1 = g^{y_1}, \ldots, u_k = g^{y_k}$. Then set $u_0 = g^b$, $w = g^b g^{x_w}$, $z = g^{x_z}$ and $h = g^{-b\alpha^*} g^{x_h}$, for random $x_w, x_z, x_h \in \mathbb{Z}_p$.

The simulator outputs the public parameters as $(g, u_0, \ldots, u_k, w, z, h)$. For the challenge keys, it sets the public key

as $pk_1 = g^a$, implicitly sets the secret key as $sk_1 = a$, and sets the internal time record as $s_{prev} = 1$. For all other keys $i = 2$ to N, it chooses a random $a_i \in \mathbb{Z}_p$, sets $pk_i = g^{a_i}$ and $sk_i = a_i$. It outputs the key information as $(pk_1, (pk_2, sk_2), \ldots, (pk_N, sk_N))$.

Queries.

When the adversary asks for a signature on message $M = M_1 M_2 \ldots M_k \in \{0,1\}^Z$, the simulator first checks the clock as $s = \mathsf{clock}()$. If $s \leq s_{prev}$ or $s \geq 2^\lambda$, it outputs $\perp$. Otherwise, it updates its time period recorder $s_{prev} := s$.

If $\alpha^* = \lceil \lg(s) \rceil$, the simulator's guess was incorrect and it aborts. Otherwise, it computes the signature by choosing random $t' \in \mathbb{Z}_p$, computing $\alpha = \lceil \lg(s) \rceil$ and

$$\begin{aligned} \sigma_2 &= g^{t'} / (g^a)^{1/(\alpha - \alpha^*)} = g^{t' - a/(\alpha - \alpha^*)} \\ \sigma_1 &= (g^a)^{\sum_{i=1}^k y_i M_i} \cdot \sigma_2^{x_w \alpha + x_z s + x_h} \cdot (g^b)^{t'(\alpha - \alpha^*)} \end{aligned}$$

and outputting (σ_1, σ_2, s).

Let us implicitly set the randomness $t = t' - a/(\alpha - \alpha^*)$ (here t' gives t the proper distribution) and we have

$$\sigma_1 = (\prod_{i=1}^k u_i^{M_i})^a \cdot (g^{x_w \alpha} z^s g^{x_h})^t \cdot g^{bt'(\alpha - \alpha^*)} \quad , \quad \sigma_2 = g^t \quad , \quad s.$$

To verify correctness, notice that we can rewrite σ_1 as follows:

$$\begin{aligned} \sigma_1 &= (\prod_{i=1}^k u_i^{M_i})^a \cdot g^{ab} \cdot (g^{x_w \alpha} z^s g^{x_h})^t \cdot g^{bt'(\alpha - \alpha^*)} \\ &\quad \cdot g^{-ab(\alpha - \alpha^*)/(\alpha - \alpha^*)} \\ &= (g^b \prod_{i=1}^k u_i^{M_i})^a (g^{x_w \alpha} z^s g^{x_h})^t \cdot g^{b\alpha(t' - a/(\alpha - \alpha^*))} \\ &\quad \cdot g^{-b\alpha^*(t' - a/(\alpha - \alpha^*))} \\ &= (u_0 \prod_{i=1}^k u_i^{M_i})^a \cdot (g^{x_w \alpha} z^s g^{x_h})^t \cdot g^{b\alpha t} \cdot g^{-b\alpha^* t} \\ &= (u_0 \prod_{i=1}^k u_i^{M_i})^a \cdot ((g^b g^{x_w})^\alpha z^s (g^{-b\alpha^*} g^{x_h}))^t \\ &= (u_0 \prod_{i=1}^k u_i^{M_i})^a \cdot (w^\alpha z^s h)^t \end{aligned}$$

Output.

Eventually, the type I adversary outputs a valid aggregate signature $\tilde{\sigma} = (\tilde{\gamma_1}, \tilde{\gamma_2}, \tilde{s})$ on some message sequence $(M_1, \ldots, M_N) \in \{0,1\}^{Z \times N}$ under public keys $(pk_1, \ldots, pk_N)$ such that $2^{\lceil \lg(q) \rceil} < \tilde{s} < 2^\lambda$. We parse each M_i in chunks as $M_{i,1} \ldots M_{i,k}$, where each chunk is ℓ bits. From the verification equation, we see that

$$e(\tilde{\gamma_1}, g) = e(\prod_{i=1}^N g^{a_i}, u_0) \cdot e(\tilde{\gamma_2}, w^{\lceil \lg(\tilde{s}) \rceil} z^{\tilde{s}} h) \cdot \prod_{j=1}^k e(\prod_{i=1}^N g^{a_i M_{i,j}}, u_j).$$

Interpreting $\tilde{\gamma_2}$ as g^t, for some $t \in \mathbb{Z}_p$, it follows from the above equation that

$$\tilde{\gamma_1} = \prod_{i=1}^N (u_0 \prod_{j=1}^k u_j^{M_{i,j}})^{a_i} (w^{\lceil lg(\tilde{s}) \rceil} z^{\tilde{s}} h)^t.$$

Let $\tilde{\alpha} = \lceil \lg(\tilde{s}) \rceil$. If $\alpha^* = \tilde{\alpha}$, then the simulator guessed correctly and, since $a_1 = a$, we have that:

$$\begin{aligned} \tilde{\gamma_1} &= \prod_{i=1}^N (g^b \prod_{j=1}^k g^{y_j M_{i,j}})^{a_i} \cdot ((g^{b + x_w})^{\tilde{\alpha}} (g^{x_z})^{\tilde{s}} (g^{-b\alpha^* + x_h}))^t \\ &= \prod_{i=1}^N (g^b \prod_{j=1}^k g^{y_j M_{i,j}})^{a_i} \cdot (g^t)^{x_w \tilde{\alpha} + x_z \tilde{s} + x_h} \\ &= g^{ab} \cdot (g^b)^{\sum_{i=2}^N a_i} \cdot (g^a)^{\sum_{j=1}^k y_j M_{1,j}} \\ &\quad \cdot \prod_{i=2}^N \prod_{j=1}^k g^{y_j M_{i,j} a_i} \cdot (g^t)^{x_w \tilde{\alpha} + x_z \tilde{s} + x_h} \\ &= g^{ab} \cdot (g^b)^{\sum_{i=2}^N a_i} \cdot (g^a)^{\sum_{j=1}^k y_j M_{1,j}} \\ &\quad \cdot g^{\sum_{i=2}^N (a_i (\sum_{j=1}^k M_{i,j} y_j))} \cdot (g^t)^{x_w \tilde{\alpha} + x_z \tilde{s} + x_h} \end{aligned}$$

The simulator computes

$$\begin{aligned} A &= (g^a)^{\sum_{j=1}^k y_j M_{1,j}} \\ B &= (g^b)^{\sum_{i=2}^N a_i} \end{aligned}$$

outputs g^{ab} computed as

$$\frac{\tilde{\gamma_1}}{A \cdot B \cdot g^{\sum_{i=2}^N (a_i (\sum_{j=1}^k M_{i,j} y_j))} \cdot (g^t)^{x_w \tilde{\alpha} + x_z \tilde{s} + x_h}}$$

If $\alpha^* \neq \tilde{\alpha}$, the simulator aborts. $\alpha^* = \tilde{\alpha}$ with probability $1/\lambda$. Therefore, if a type I adversary can break this scheme with probability of ϵ, then the simulator can solve the CDH problem with probability of at least ϵ/λ. $\square$

4.1.2 Type II Adversary

Proof of Lemma 4.2 appears just above. It covers the case where the adversary forges with a time period that is "too high" and thus is similar to [18]. Our proof of the following lemma is more interesting, as we no longer have a chameleon hash function to help answer the adversary's signing queries and thus must find a new strategy. We break the message into ℓ-bit chunks (instead of chameleon hashing it). This allows us to keep all portions of the signed message "out from under" any fresh randomness in the signature, which enable aggregation. In the proof, the simulator makes three guesses: $s^* \in [1, 2^{\lceil \lg(q) \rceil}]$, $M' \in [0, 2^\ell - 1]$ and $\beta^* \in [1, k]$. These individual guesses represent a single guess that the β^*th chunk of the forgery message will differ from the value M' which will be the β^*th chunk of the message that the adversary asks to sign at time period s^*. We can then use techniques by Boneh and Boyen [10] to simulate.

LEMMA 4.3. *Suppose the (t', ϵ')-CDH assumption holds in $\mathbb{G}$ of prime order p. Then the aggregate signature scheme above is (t, q, N, ϵ)-secure against type II existential forgery under an adaptive chosen message attack provided that*

$$\epsilon \geq \epsilon' \cdot (2^{\ell+1} \cdot q \cdot k) \quad , \quad t \leq t' - \Theta(T(N + q + k))$$

where q is the number of signing queries, $Z = \ell \cdot k \in O(\lambda)$ is the message length, $2^\lambda < p$, and T is the maximum time for an exponentiation in $\mathbb{G}$ or $\mathbb{G}_T$.

PROOF. Given a CDH challenge (g, g^a, g^b), proceed as:

Setup.

The simulator begins by making three guesses. First, it guesses the time period s^* in the range 1 to $2^{\lceil \lg(q) \rceil}$ which the adversary will use to forge. Second, it guesses an ℓ-bit message chunk M'. Third, it guesses a special message chunk β^* in $[1, k]$. These individual guesses represent a single guess that the β^*th chunk of the forgery message will differ from the value M' which will be the β^*th chunk of the message that the adversary asks to sign at time period s^*.

Next, choose random values $x_0, \ldots, x_k \in \mathbb{Z}_p$ and set $u_0 = g^{-bM'} g^{x_0}$, $u_{\beta^*} = g^b$ and $u_i = g^{x_i}$ for all other i from 1 to k. Then choose random $x_w, x_z, x_h \in \mathbb{Z}_p$ and set $w = g^{x_w}$, $z = g^b g^{x_z}$ and $h = g^{-bs^*} g^{x_h}$.

The simulator outputs the public parameters as $(g, u_0, \ldots, u_k, w, z, h)$. For the challenge keys, it sets the public key as $pk_1 = g^a$, implicitly sets the secret key as $sk_1 = a$, and sets the internal time record as $s_{prev} = 1$. For all other keys $i = 2$ to N, it chooses a random $a_i \in \mathbb{Z}_p$, sets $pk_i = g^{a_i}$ and $sk_i = a_i$. It outputs the key information as $(pk_1, (pk_2, sk_2), \ldots, (pk_N, sk_N))$.

Queries.

When the adversary asks for a signature on message $M = M_1 M_2 \ldots M_k \in \{0,1\}^Z$, the simulator first checks the clock as $s = \mathsf{clock}()$. If $s \leq s_{prev}$ or $s \geq 2^\lambda$, it outputs $\perp$. Otherwise, it updates its time period recorder $s_{prev} := s$. There are now two ways the simulator will proceed.

If $s = s^*$, then check that $M_{\beta^*} = M'$. If this is not true, then the simulator's guess was incorrect and it must abort. Otherwise, let $I := \{1, \ldots, k\} - \{\beta^*\}$. It chooses a random $t \in \mathbb{Z}_p$ and sets

$$\sigma_1 = (g^a)^{x_0 + \sum_{i \in I} x_i M_i} \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \quad , \quad \sigma_2 = g^t \quad , \quad s.$$

To verify correctness, observe that we can rewrite σ_1 as follows:

$$\begin{aligned} \sigma_1 &= (g^{ab})^{-M' + M'} (g^a)^{x_0 + \sum_{i \in I} x_i M_i} \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \\ &= (g^{-bM + x_0} g^{bM'} g^{\sum_{i \in I} x_i M_i})^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \\ &= (u_0 \prod_{i=1}^{k} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \end{aligned}$$

If $s \neq s^*$, then choose random $t' \in \mathbb{Z}_p$ and compute the value $V = g^{t'} / (g^a)^{(M_{\beta^*} - M')/(s - s^*)} = g^{t' - a(M_{\beta^*} - M')/(s - s^*)}$. Let $I := \{1, \ldots, k\} - \{\beta^*\}$. Output (σ_1, σ_2, s) where

$$\begin{aligned} \sigma_2 &= V \\ \sigma_1 &= (g^a)^{x_0 + \sum_{i \in I} x_i M_i} \cdot \sigma_2^{x_w \lceil \lg(s) \rceil + x_z s + x_h} \cdot (g^b)^{t'(s - s^*)} \end{aligned}$$

Let us implicitly set the randomness $t = t' - a(M_{\beta^*} - M')/(s - s^*)$ (here t' gives t the proper distribution) and we have

$$\begin{aligned} \sigma_1 &= (g^{x_0} \prod_{i \in I} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^b)^{t'(s - s^*)} \\ \sigma_2 &= g^t \end{aligned}$$

To verify correctness, notice that we can rewrite σ_1 as:

$$\begin{aligned} \sigma_1 &= (g^{ab})^{(M_{\beta^*} - M')} \cdot (g^{x_0} \prod_{i \in I} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \\ &\quad \cdot (g^b)^{t'(s - s^*)} \cdot (g^{-ab})^{(M_{\beta^*} - M')} \\ &= (g^{-bM' + x_0} g^{bM_{\beta^*}} \prod_{i \in I} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \\ &\quad \cdot (g^b)^{t'(s - s^*)} \cdot (g^{-ab})^{(M_{\beta^*} - M')} \\ &= (u_0 \prod_{i=1}^{k} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} g^{x_z s} g^{x_h})^t \cdot (g^{b(s - s^*)})^t \\ &= (u_0 \prod_{i=1}^{k} u_i^{M_i})^a \cdot (w^{\lceil \lg(s) \rceil} z^s h)^t \end{aligned}$$

Output.

Eventually, the type II adversary outputs a valid aggregate signature $\tilde{\sigma} = (\tilde{\gamma_1}, \tilde{\gamma_2}, \tilde{s})$ on a message sequence $(M_1, \ldots, M_N) \in \{0,1\}^{Z \times N}$ under public keys $(pk_1, \ldots, pk_N)$ such that $0 < \tilde{s} \leq 2^{\lceil \lg(q) \rceil} \leq 2q$. We parse each M_i in chunks as $M_{i,1} \ldots M_{i,k}$, where each chunk is ℓ bits. From the verification equation, we see that

$$e(\tilde{\gamma_1}, g) = e(\prod_{i=1}^{N} g^{a_i}, u_0) \cdot e(\tilde{\gamma_2}, w^{\lceil \lg(\tilde{s}) \rceil} z^{\tilde{s}} h) \cdot \prod_{j=1}^{k} e(\prod_{i=1}^{N} g^{a_i M_{i,j}}, u_j).$$

If $s^* = \tilde{s}$ and $M_{1,\beta^*} \neq M'$, then the simulator guessed correctly. In this case, let $J := \{1, \ldots, k\} - \{\beta^*\}$. Interpreting $\tilde{\gamma_2}$ as g^t, for some $t \in \mathbb{Z}_p$, it follows from the above equation that

$$\begin{aligned} \tilde{\gamma_1} &= (g^{-bM' + x_0})^{\sum_{i=1}^{N} a_i} \cdot g^{b \sum_{i=1}^{N} a_i M_{i,\beta^*}} \\ &\quad \cdot g^{\sum_{i=1}^{N} a_i \sum_{j \in J} x_j M_{i,j}} \\ &\quad \cdot ((g^{x_w})^{\lceil \lg(\tilde{s}) \rceil} (g^{b + x_z})^{\tilde{s}} (g^{-bs^* + x_h}))^t \\ &= (g^{-bM' + x_0})^{\sum_{i=1}^{N} a_i} \cdot g^{b \sum_{i=1}^{N} a_i M_{i,\beta^*}} \\ &\quad \cdot g^{\sum_{i=1}^{N} a_i \sum_{j \in J} x_j M_{i,j}} \\ &\quad \cdot g^{t(x_w \lceil \lg(\tilde{s}) \rceil + x_z \tilde{s} + x_h)} \\ &= g^{ab(M_{1,\beta^*} - M')} \cdot g^{ax_0} \cdot (g^{-bM' + x_0})^{\sum_{i=2}^{N} a_i} \\ &\quad \cdot g^{b \sum_{i=2}^{N} a_i M_{i,\beta^*}} \\ &\quad \cdot g^{\sum_{i=1}^{N} a_i \sum_{j \in J} x_j M_{i,j}} \cdot g^{t(x_w \lceil \lg(\tilde{s}) \rceil + x_z \tilde{s} + x_h)} \\ &= g^{ab(M_{1,\beta^*} - M')} \cdot g^{a(x_0 + \sum_{j \in J} x_j M_{1,j})} \\ &\quad \cdot g^{b(\sum_{i=2}^{N} a_i (M_{i,\beta^*} - M'))} \\ &\quad \cdot g^{\sum_{i=2}^{N} a_i (x_0 + \sum_{j=1}^{k} x_j M_{i,j})} \cdot g^{t(x_w \lceil \lg(\tilde{s}) \rceil + x_z \tilde{s} + x_h)} \end{aligned}$$

The simulator computes

$$\begin{aligned} A &= (g^a)^{x_0 + \sum_{j \in J} x_j M_{1,j}} \\ A' &= g^{\sum_{i=2}^{N} a_i (x_0 + \sum_{j=1}^{k} x_j M_{i,j})} \\ B &= (g^b)^{\sum_{i=2}^{N} a_i (M_{i,\beta^*} - M')} \end{aligned}$$

and outputs g^{ab} as

$$\left(\frac{\tilde{\gamma_1}}{B \cdot A \cdot A' \cdot (g^t)^{x_w \lceil \lg(\tilde{s}) \rceil + x_z \tilde{s} + x_h}} \right)^{\frac{1}{M_{1,\beta^*} - M'}}.$$

Otherwise, the simulator aborts. The probability it does not abort at any point during the simulation is

$$\frac{1}{k} \cdot \frac{1}{2^{\lceil \lg q \rceil}} \cdot \frac{1}{2^{\ell}} = \frac{1}{k \cdot 2^{\lceil \lg q \rceil} \cdot 2^{\ell}} \geq \frac{1}{k \cdot 2q \cdot 2^{\ell}}$$

Therefore, if a type II adversary can break this scheme with probability of ϵ, then the simulator can solve the CDH problem with probability of at least $\epsilon/(2^{\ell+1} \cdot q \cdot k)$. □

5. DISCUSSION

We discuss features and extensions of the construction.

Aggregating Aggregates. In our scheme, anyone can combine two aggregate signatures into a single aggregate signature simply by multiplying them together, provided that the time periods match. This is also a useful property of other existing schemes [11, 16, 6].

Self-Aggregation and Multiple Signatures per Period. In Section 4, a signer can only issue one signature per time period, however, this requirement can be relaxed if the size of a user's public key is allowed to grow in proportion to the total number of messages to be signed in any time period. The technique is based on the ability to self-aggregate. A user who wishes to sign at most j messages per time period must select a public key $pk = (g^{a_1}, \ldots, g^{a_j})$. To sign δ ($\leq j$) messages $M_1, M_2, \ldots, M_\delta$ where $M_i = M_{i,1} M_{i,2} \ldots M_{i,k}$ in the current time period s, the signer can the generate an aggregate signature on all δ messages, with randomness $t \in \mathbb{Z}_p$, as:

$$\sigma_1 = (\prod_{j=1}^{\delta} (u_0 \prod_{i=1}^{k} u_i^{M_{j,i}})^{a_j}) \cdot (w^{\lceil lg(s) \rceil} z^s h)^t, \quad \sigma_2 = g^t, \quad s.$$

Most Signing Work Can Be Done Offline. If the signer knows a future time period s during which she wants to issue a signature, then she can precompute the values $(w^{\lceil \lg(s) \rceil} z^s h)^t$ and g^t. If we let messages chunks be bits and the signer stores her secret key in the form $u_0^a, u_1^a, \ldots, u_Z^a$, then she can compute the signature once it is known using $Z+1$ multiplications to get $(u_0 \prod_{i=1}^{Z} u_i^{M_i})^a$ and then one final multiplication to obtain

$$(u_0 \prod_{i=1}^{Z} u_i^{M_i})^a (w^{\lceil \lg(s) \rceil} z^s h)^t.$$

This might help a lower-resource device get its signatures off quickly once the message value becomes known.

Batch Verification. A batch verification algorithm [5] takes as input n signatures on n messages from n users and outputs 1 if all individual signatures verify (with probability 1) and 0 otherwise (with probability $1 - 2^{-L}$ for security parameter L.)

The same signatures we aggregated in Section 4 also batch verify. The batching algorithm works even for different signers on different messages at different time periods. It requires only $k+5$ pairings for N signatures, where k is the security parameter from before which in practice could be 5. Let L be a security parameter, which in practice could be 80. It works as follows:

Batch$((pk_1, M_1, \sigma_1), \ldots, (pk_N, M_N, \sigma_N))$ The batch verification algorithm parses each signature $\sigma_i = (\sigma_{1,i}, \sigma_{2,i}, s_i)$ and checks that all $0 < s_i < 2^\lambda$. If this is false, it rejects. Let $M_i = M_{i,1} M_{i,2} \ldots M_{i,k}$, where each division is ℓ bits. The algorithm extracts $g^{a_i} \in pk_i$ and batch verifies the signatures by checking the group membership of all $(\sigma_{1,i}, \sigma_{2,i})$ values, choosing $r_1, \ldots, r_N \in \{0,1\}^L$ and testing that:

$$\begin{aligned} e(\prod_{i=1}^{N} \sigma_{1,i}^{r_i}, g) = {} & e(\prod_{i=1}^{N} g^{a_i r_i}, u_0) \cdot e(\prod_{i=1}^{N} \sigma_{2,i}^{r_i \lceil \lg(s_i) \rceil}, w) \\ & \cdot e(\prod_{i=1}^{N} \sigma_{2,i}^{r_i s_i}, z) \cdot e(\prod_{i=1}^{N} \sigma_{2,i}^{r_i}, h) \\ & \cdot \prod_{j=1}^{k} e(\prod_{i=1}^{N} g^{a_i r_i M_{i,j}}, u_j) \end{aligned}$$

THEOREM 5.1. *The above algorithm is a batch verifier for the Section 4 signatures with error* 2^{-L}.

In many applications, signatures may be streaming into a collector, who batch verifies them, and then aggregates the most interesting of them for storage or transmission purposes. Even existing random oracle schemes do not provide this functionality. The full aggregate signatures of [11, 6] do not batch verify for different signers on different messages and the (synchronized) signatures of [16] do not batch verify across synchronization values. The sequential signatures of [23] also require $\Omega(N)$ pairings to verify N signatures, as they employ $O(\lambda)$ different group elements *per signer*.

One can extend this algorithm to batch verify a group of aggregate signatures in a straightforward manner. Thus, a central database could receive a group of aggregate signatures at various times during the day and then quickly batch verify them all together.

Better Efficiency in the Random Oracle Model. In Appendix A, we provide a synchronized aggregate signature construction in the random oracle model which is strictly more efficient than our standard model construction, but also has properties which may make it more desirable for some applications than existing random oracle schemes. Our Section 4 scheme has $O(\lambda)$ elements in the public parameters, which results in verification times of the same order (although independent of N.) In our random oracle model scheme, we require only 6 elements in the public parameters, and both our signatures and aggregate signatures can be verified using at most 4 pairings. Our random oracle model scheme also batch verifies efficiently.

In contrast, the Boneh et al. [11] scheme requires $N+1$ pairings to verify an aggregate signature from N signers on N different messages. In Gentry and Ramzan's synchronized scheme [16], verifications require only 3 pairings. However, all prior full or synchronized aggregate signatures [11, 16, 6] require a hash function $H : \{0,1\}^* \to \mathbb{G}$, where $\mathbb{G}$ is a bilinear group, where as our random-oracle construction only requires a hash $H' : \{0,1\}^* \to \mathbb{Z}_p$. Since there are some candidate elliptic curve implementations of bilinear groups where efficient algorithms for hashing into $\mathbb{G}$ are unknown or are rather costly [15], our constructions (both in and out of the random oracle model) allow for a potentially wider set of implementation options and possibly significantly faster verification times.

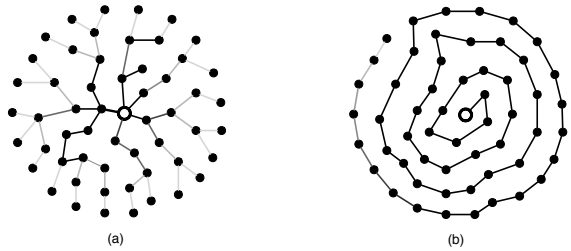

Figure 2: Sensor networks with a single collector. Line shadings indicate the total message bandwidth on each link. Figure (*a*) shows a typical routing pattern that might be used in a distributed network; by aggregating at each hop, synchronized aggregate signatures ultimately reduce the total signature bandwidth to approximately the size of a single signature. Figure (*b*) illustrates a contrived routing structure necessary to achieve the same result with a sequential aggregate signature.

6. APPLICATIONS

We now briefly describe some of the applications for synchronized aggregate signatures.

Reducing bandwidth in sensor and ad-hoc networks. Sensor networks [2, 22] consist of limited, often battery-powered devices that collect measurements over a wide area and route them to one or more central base stations for collection. In some applications, it may be necessary to cryptographically authenticate these measurements in order to mitigate the possibility of false data being injected. This is particularly important when the authenticity of the data being collected must be assured, e.g., patient vital signs in a hospital setting [21] or status messages in a vehicular communication network [12].

While various solutions to the problem of authenticating sensor messages have been proposed (e.g., [28, 27]), most authenticate only on a hop by hop basis or provide only temporary (non-repudiable) security (for example, using MACs). Unfortunately, this may not be sufficient to protect communications in a sensor network, where it is relatively easy to compromise intermediate nodes and inject false data.

Digital signatures offer better security properties, but can add significant bandwidth overhead. This is problematic given that sensors often run on battery power and must minimize radio communications. Furthermore, the extra transmission requirements fall disproportionately on those nodes closest to the base station. Figure 2 (*a*) shows a typical routing configuration for such a network.

Synchronized aggregate signatures may reduce the bandwidth requirements that message signing imposes on a network. Rather than carry all signature data, intermediate routing nodes can perform signature aggregation at any point where multiple signatures must be routed towards the collector. Of course, for aggregation to work in our scheme we require that many nodes sign their messages under the same state. This can easily be achieved by deriving state from a loosely synchronized clock. Indeed this requirement is not unreasonable given that many networks already carry time synchronization messages [22]. Since aggregate signatures can themselves be aggregated in our scheme (Section 5), the base station of Figure 2 only needs to store a single final aggregate. In Figure 2 (*b*), we contrast this with the impractical routing pattern necessary to achieve the same result using *sequential* aggregate signatures [23], which require that messages be aggregated in a sequential path.

We note that unlike systems such as TESLA [27], which also relies on synchronized clocks for authentication, loss of clock synchronization in our approach does not compromise security. Should nodes become out-of-synch, this will only reduce the *efficiency* of the aggregation process until such time as synchronization can be achieved. Of course, it is necessary to prevent individual signers from *reusing* state. This is possible to avoid through correct system design.

Software authentication. Mobile and embedded operating systems are increasingly using code signing to ensure that only legitimate binaries have privileges to run on a device. In constrained systems where storage is at a premium, the additional storage cost of these signatures may be significant. This is particularly true in systems that contain many small signed binaries, e.g., dynamic libraries.

For applications where signature verification can be performed all at once, for instance at boot time, it might be feasible to sign all binaries under a single signature. However, the contents of a system may change periodically (due to software patches and installation of new software), which would not be supported by this approach. We propose to instead reduce the signature overhead by aggregating signatures using a synchronized scheme. Due to the dynamic nature of the aggregation process, new applications and libraries can be dynamically installed on the system as necessary, and the aggregate can be periodically updated.

In a synchronized aggregate signature it is necessary that all signatures under aggregation share the same state value. While a synchronized clock does not seem appropriate here, software version numbers may offer an alternative source for signature state. Signatures on all binaries with revision number 1 could be aggregated together, as could signatures with software revision 2 and so on. If there is a significant degree of overlap, this could result in meaningful savings.

7. CONCLUSION

We presented the first aggregate signature construction in the standard model that does not require any form of interaction among signers to generate. It requires that signers have access to a synchronized clock and only signatures from the same period can be aggregated. Our construction is practical and based on the Computational Diffie-Hellman assumption. It is also the first (non-sequential) aggregate scheme where the underlying signatures can be batch verified across different signers, messages and time periods. Thus, it is a good candidate for a variety of communication applications where routing flexibility, speed and low bandwidth are needed. We discussed the benefits of using this approach over sequential aggregation or symmetric authentication in sensor network and software authentication applications.

It remains open to construct a practical aggregation scheme in the standard model without (1) timing or interactive restrictions or (2) requiring that each user be able to prove knowledge of her secret key. We are not certain whether or not the former, in particular, is possible. If it is not, it would be interesting to prove this. It is also open to explore other relaxations of the full aggregation model.

Acknowledgments

The authors are grateful to Brent Waters and the CCS 2010 anonymous reviewers for their helpful comments. The authors were supported by NSF Grant CNS-0716142 and Department of Homeland Security Grant 2006-CS-001-000001-02 (subaward 641). In addition, Matthew Green was supported by NSF Grant CNS-1010928 and Susan Hohenberger was supported by a Microsoft New Faculty Fellowship and a Google Research Award.

8. REFERENCES

[1] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: New definitions, constructions and applications, 2010. Full version available at `http://eprint.iacr.org`.

[2] I. F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[3] Ali Bagherzandi and Stanislaw Jarecki. Identity-Based Multi-Signatures based on RSA. In *PKC '10*, volume 6056 of LNCS, pages 480–498, 2010.

[4] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS '04*, pages 186–195, 2004.

[5] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of LNCS, pages 236–250, 1998.

[6] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP '07*, volume 4596 of LNCS, pages 411–422, 2007.

[7] Mihir Bellare and Gregory Neven. Identity-Based Multi-signatures from RSA. In *CT-RSA '07*, volume 4377 of LNCS, pages 145–162, 2007.

[8] 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 (CCS)*, pages 276–285, 2007.

[9] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing, 2010. Full version available at `http://www.cc.gatech.edu/~amoneill/bgoy.html`.

[10] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027, pages 223–238, 2004.

[11] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT '03*, volume 2656 of LNCS, pages 416–432, 2003.

[12] Car 2 Car. Communication consortium. `http://car-to-car.org`.

[13] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[14] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23:224–280, 2010.

[15] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[16] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *Public Key Cryptography '06*, volume 3958 of LNCS, pages 257–273, 2006.

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

[18] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT '09*, volume 5479 of LNCS, pages 333–350, 2009.

[19] Jung Yeon Hwang, Dong Hoon Lee, and Moti Yung. Universal forgery of the identity-based sequential aggregate signature scheme. In *ASIACCS '09*, pages 157–160, 2009.

[20] Stephen Kent, Charles Lynn, and Karen Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[21] JeongGil Ko, Tia Gao, Richard Rothman, and Andreas Terzis. Wireless sensing systems in clinical environments: Improving the efficiency of the patient monitoring process. *IEEE Engineering in Medicine and Biology (EMB) Magazine*, 29(2):103–109, 2010.

[22] Chieh-Jan Mike Liang, Jie Liu, Liqian Luo, Andreas Terzis, and Feng Zhao. RACNet: A high-fidelity data center sensing network. In *ACM Conference on Embedded Networked Sensor Systems (SenSys) '09*, pages 15–28, 2009.

[23] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT '06*, volume 4004 of LNCS, pages 465–85, 2006. Full version at `http://cseweb.ucsd.edu/~hovav/dist/agg-sig.pdf`.

[24] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT '04*, volume 3027 of LNCS, pages 74–90, 2004.

[25] David Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.

[26] Gregory Neven. Efficient sequential aggregate signed data. In *EUROCRYPT '08*, volume 4965 of LNCS, pages 52–69, 2008.

[27] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *NDSS '01*, pages 35–46, February 2001.

[28] Harald Vogt. Exploring message authentication in sensor networks. In *Security in Ad-hoc and Sensor Networks*, volume 3313 of LNCS, pages 19–30. Springer, 2005.

[29] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05*, volume 3494, pages 320–329, 2005.

APPENDIX

A. A RANDOM ORACLE CONSTRUCTION WITH BETTER EFFICIENCY

We provide an optimized version of the synchronized aggregate construction in Section 4. The primary savings are a reduction in the size of the public key and a quicker verification algorithm. We analyze the specifics of the improvements after presenting the scheme.

Setup(1^λ).

The setup algorithm selects a bilinear group $\mathbb{G}$ of prime order $p > 2^\lambda$. Select a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function treated as a random oracle. It chooses random group elements $g, u, v, w, z, h \in \mathbb{G}$. It outputs the public parameters as $\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, H, g, u, v, w, z, h)$. As before, we assume all parties have access to a function $\mathsf{clock}()$ that on no input, returns the current time period as an element in $\mathbb{Z}$.

KeyGen($1^\lambda, \mathsf{pp}$).

The key generation algorithm takes as input the parameters pp and selects a random $a \in \mathbb{Z}_p$. It outputs the public key as $\mathrm{PK} = (\mathsf{pp}, g^a)$ and the secret key as $\mathrm{SK} = (\mathsf{pp}, a)$. It also initializes s_{prev} to be zero.

Sign($\mathrm{SK}, M \in \{0,1\}^*, s$).

The signer obtains $s = \mathsf{clock}()$. If $s \leq s_{prev}$ or $s \geq 2^\lambda$, then abort. Otherwise, record the current time period as $s_{prev} := s$. The signing algorithm selects a random $t \in \mathbb{Z}_p$ and then outputs a signature on M under key SK and time period s as:

$$\sigma_1 = (vu^{H(M)})^a \cdot (w^{\lceil lg(s) \rceil} z^s h)^t, \quad \sigma_2 = g^t, \quad s.$$

Verify($\mathrm{PK}, M, \sigma = (\sigma_1, \sigma_2, s)$).

The verification algorithm first makes sure that $0 < s < 2^\lambda$. If this is false, then it rejects. It verifies the signature by checking that

$$e(\sigma_1, g) = e(g^a, vu^{H(M)}) \cdot e(\sigma_2, w^{\lceil \lg(s) \rceil} z^s h).$$

Aggregate($(pk_1, M_1, \sigma_1), \ldots, (pk_N, M_N, \sigma_N)$).

Parse σ_1 as $(\sigma_{1,1}, \sigma_{1,2}, s)$. The aggregation algorithm checks that $\mathsf{Verify}(pk_i, M_i, \sigma_i) = 1$ and that s is the third element of σ_i for $i = 1$ to N. If any check fails, it outputs $\perp$. Otherwise, it parses σ_i as $(\sigma_{i,1}, \sigma_{i,2}, s)$ and computes

$$\gamma_1 = \prod_{i=1}^{N} \sigma_{i,1} \quad , \quad \gamma_2 = \prod_{i=1}^{N} \sigma_{i,2}$$

The aggregate signature is output as (γ_1, γ_2, s).

AggVerify($(pk_1, \ldots, pk_N), (M_1, \ldots, M_N), \sigma$).

The verification algorithm checks that $0 < s < 2^\lambda$. If this is false, it rejects. The algorithm extracts $g^{a_i} \in pk_i$ and verifies the signature by checking that

$$e(\gamma_1, g) = e(\prod_{i=1}^{N} g^{a_i}, v) \cdot e(\prod_{i=1}^{N} g^{a_i H(M_i)}, u) \cdot e(\gamma_2, w^{\lceil \lg(s) \rceil} z^s h).$$

THEOREM A.1. *Suppose the (t', ϵ')-CDH assumption holds in group $\mathbb{G}$ of prime order p. Then the aggregate signature scheme above is $(t, q_s, q_H, N, \epsilon)$-secure against existential forgery under an adaptive chosen message attack provided that*

$$\epsilon \geq 2\epsilon' \cdot \max(2q_s \cdot q_H, \lambda) \quad , \quad t \leq t' - \Theta(T(N + q_s))$$

where q_s is the number of signing queries, q_H is the number of random oracle queries, $2^\lambda < p$, and T is the maximum time for an exponentiation in $\mathbb{G}$ or $\mathbb{G}_T$.

Proof of this theorem follows techniques of the prior proof and appears in the full version of this work [1].

Efficiency Discussion. The public parameters are 6 elements in $\mathbb{G}$; public keys are 1 element in $\mathbb{G}$, and signatures are two elements in $\mathbb{G}$ plus a small integer. Aggregate verification of n signatures requires only 4 pairing operations and, to our knowledge, this is the first aggregate construction in bilinear groups (except for Section 4) which does not require hashing a string into $\mathbb{G}$. For some choices of elliptic curve implementations, efficient hash functions are not known and in some others, they are rather expensive (comparable to the cost of a pairing itself). Because our construction requires a simple hash into $\mathbb{Z}_p$ instead of a more complex hash into $\mathbb{G}$, we can allow for a wider class of implementation options as well as significantly improve the verification time in some implementations.