A Provably Secure Short Transitive Signature Scheme from Bilinear Group Pairs

Siamak Fayyaz Shahandashti[†], Mahmoud Salmasizadeh[‡], and Javad Mohajeri[‡]

[†] School of Electrical Engineering, Sharif University of Technology, Tehran, Iran
siamak@ee.sharif.edu
http://ee.sharif.edu/~siamak
[‡] Electronic Research Center, Sharif University of Technology, Tehran, Iran
{salmasi, mohajer}@sharif.edu

Abstract. We present a realization of the transitive signature scheme based on the algebraic properties of bilinear group pairs. The scheme is proven secure, i.e. transitively unforgeable under adaptive chosen message attack, assuming hardness of the computational co-Diffie-Hellman problem in bilinear group pairs and the security of the underlying standard signature scheme under *known* message attack. Our scheme mostly conforms to previously designed schemes of Micali-Rivest and Bellare-Neven in structure; yet there are two contributions: firstly, we take advantage of bilinear group pairs which were previously used by Boneh, Lynn, and Shacham to build short signature schemes. Secondly, we show that a slight modification in previous definitions of the transitive signature relaxes the security requirement for the underlying standard signature from being secure under chosen message attack to being secure under known message attack; thus shorter and more efficient signatures can be chosen for the underlying standard signature. These two facts eventually yield to short transitive signatures with respect to both node and edge signature size.

1 Introduction

The concept of signature schemes with algebraic properties, later called *homomorphic* [12] or *algebraic* [11] signatures, was first introduced by Rivest in a series of talks [18]. These schemes allow an arbitrary entity to forge signatures on certain messages. Rivest mentioned that algebraic properties must not always be considered as a security threat for cryptosystems. For example, the multiplicative property of RSA function can be advantageous in certain applications. He also presented two design instances of such signature schemes: the *prefix aggregation signature scheme* and the *transitive signature scheme*.

A transitive signature is a scheme for signing vertices and edges of a dynamically growing, transitively closed graph. Transitive closure is the property of including any

* This research was supported in part by Iran Telecommunication Research Center (ITRC) grant #T/500/3649 through the School of Electrical Engineering, Sharif University of Technology.

* Please see the the Errata appendix at the end.

Originally published in C. Blundo and S. Cimato (Eds.): SCN 2004, LNCS 3352, pp.60–76, Springer, 2005.

edge if there is a path between two vertices of the two ends of the edge. Depending on the graph, the transitive signature can be directed or undirected. The problem of finding a directed transitive signature is still a challenging open problem. Therefore, the term "transitive signature" is now being used in literature in case of "undirected transitive signature". We will also use this notation through the paper.

A transitive signature has the property that *everyone* can forge a valid signature on the edge *AC* of a graph, knowing the signatures of two edges *AB* and *BC*. This *everyone* does not need to have knowledge of the secret key at all. He/She just knows the public information. Since the graph itself is transitively closed, this property cannot be counted as a deficiency in security. Furthermore, Rivest showed that this property can "provide efficiency for prover and verifier" in comparison with the use of standard signatures [18]. One obvious advantage is that by using a transitive signature, one must sign only $O(n)$ edges of a graph with size n in case of $O(n^2)$ standard signings [15].

To achieve another advantage, a transitive signature must have the property that the *composed* signature on the edge *AC* should be indistinguishable from the signature that could have been produced by the *original* signer on it. This allows the receiver of the signatures to reveal no extra information when presenting the composed signature on the edge *AC* to a third person. A distinguishable composed signature at least bears the information that some other node *B* is between nodes *A* and *C*.

Micali and Rivest introduced the first provably secure (undirected) transitive signature scheme in [15]. Their scheme's security is based on the infeasibility of solving the discrete logarithm problem. Later, Bellare and Neven introduced new schemes whose security proofs were based on the hardness of factoring and on the security of RSA under one-more-inversion [2]. More new schemes based on gap Diffie-Hellman groups also appear in the recent full version of their paper (See [3]).

The security of many recently designed cryptosystems is based on hardness of the computational Diffie-Hellman and co-Diffie-Hellman problems in the so called Gap-Diffie-Hellman (GDH) groups. A GDH group is a group in which decision Diffie-Hellman (DDH) problem is easy, while computational Diffie-Hellman (CDH) problem is hard to solve. Signature schemes with provable security, both in random oracle and in standard model, are designed by Boneh et al. (See [7] and [8].) using bilinear maps in GDH groups. Also many other encryption schemes, signatures (plain, blind, proxy, ring, undeniable, group ...), key agreement protocols (plain, authenticated, group ...), access control, etc. have been constructed based on bilinear maps (See [1] and [14]).

A bilinear map (See [6] for introduction.), also called *pairing*, is a mapping between groups in a way that is "consistent with the group structure of its arguments" [8]. In simple words, it is a mapping that has the linearity property with respect to both its arguments, i.e. there exists three operations $\circ$, $\bullet$, and $*$ such that for every g, h, x, and y we have

$$e(g \circ h, x) = e(g, x) * e(h, x) \text{ and } e(g, x \bullet y) = e(g, x) * e(g, y).$$

This property yields to some useful algebraic properties. Joux and Nguyen showed that an efficiently-computable bilinear map e provides a polynomial time algorithm for solving the decision co-Diffie-Hellman problem [13].

In this paper, we construct a transitive signature from bilinear maps and prove it secure under conventional security assumptions of such maps.

Our Contributions: The main valuable property of our design of the transitive signature scheme is that the signature size on a graph is *shorter* than those of previously designed transitive signatures. Short transitive signatures are useful to shorten the total graph signature size, and this will be vital especially when the size of the graph itself grows too big. It is also apparent that this occurs in many applications where it is needed to transmit a big graph, as graphs themselves are used to simplify understanding large amounts of information.

Our transitive signature is shorter than previous ones in two ways. Firstly, since we use bilinear group pairs to sign edges of a graph in our design as Boneh et al. did to design short signatures, all the discussions on how short the signature could be are still applicable here for edge signatures. For the detailed information, refer to the original paper [7]. Secondly, we propose a slightly modified new definition for transitive signatures, which relaxes the choice of the underlying signature scheme, making it more efficient in sense of signing and verification cost as well as signature length than many conventional signatures. This fact makes the signature on the nodes of the graph shorter and more efficient. We achieve this by showing that the security requirement for the underlying standard signature in our design is just being secure under *known* message attack, rather than adaptive ([3], [15]) or non-adaptive [19] *chosen* message attack for other schemes. Since both edge and node signature size are made shorter than previous schemes, eventually a very short transitive signature on the whole graph is resulted in!

Organization of the Paper: Section 2 is allotted to notations and definitions: In 2.1 we fix the notations we will use through the paper; 2.2 fetches the definitions of the transitive signature schemes and their security. In 2.3 we define the bilinear maps, bilinear group pairs, and their security assumptions. Finally in Section 3 a transitive signature is built using bilinear maps and also proven secure. The concluding remarks, acknowledgements, and references are followed then as usual.

2 Notations and Definitions

A review of notations and definitions used in the paper follows. We will give a mixture of the definitions of transitive signatures presented in [15], [2], and [16], which are all fundamentally the same. Then we will go through bilinear maps and define *bilinear group pairs*, as in [8].

2.1 Notations

All graphs in this paper are undirected. We will use small letters such as i, j, k for the nodes and a concatenation of two small letters (which are the two endpoints of an edge) such as ij, jk for the (undirected) edges of the graph. We will use σ for all signatures on both nodes and edges. The signature on a node i is shown as σ_i and a signature on an edge ij is shown as σ_{ij}. The expression

$$s \xleftarrow{R} S$$

means that a member s is randomly chosen from set S. The italic letter Z represents the set of integer numbers and hence

$$Z_p \text{ and } Z_p^*$$

are used for the additive and multiplicative groups modulo p. We also show the group operation with two operands g and h by $g \cdot h$ or simply by gh (as in multiplicative groups), and by g / h we mean the group operation done with the first operand and inverse of the second one (also, as in multiplicative groups).

2.2 Transitive Signature Schemes

Since, as formerly said, transitive signatures are schemes to authenticate transitively closed graphs. We first review the definition of transitive closure property in graphs.

Transitive Closure of a Graph [15]: Transitive closure of a graph $G = (V, E)$ is the graph $G^* = (V, E^*)$, such that (i,j) is an edge of G^* if and only if there is a path from i to j in G. If $G = G^*$, then we say that the graph G is transitively closed.

Standard Signature Scheme [10]: As a well-known definition, a standard signature scheme is a tuple of three algorithms $SS = (SKeyGen, SSig, Sverify)$ for key generation, signing and verifying. A pair of public and secret keys are generated as $(SPK, SSK) \leftarrow SKeyGen(1^k)$ and the signature is generated as $\sigma \leftarrow SSign(SSK, m)$ for a message m and verified valid as $\texttt{true} \leftarrow SVerify(SPK, m, \sigma)$. The signature is said to be secure against known message attack if no polynomial time adversary can forge a valid signature on a new message, knowing a list of message-signature pairs for some random messages, except with negligible probability in the security parameter k. Here, new message means a message not in the list the adversary is provided with. We denote the probability that adversary F' succeeds in forging a new message-signature pair for the standard signature SS through a known message attack by

$$Adv_{SS,F'}^{uf-kma}(k).$$

We also call the maximum advantage among all adversaries, polynomial time in k, the *insecurity function* of the standard signature SS through a known message attack and denote it by

$$InSec_{SS}^{uf-kma}(k).$$

SS is called secure under known message attack if and only if this function decreases faster than any polynomial in k.

Transitive Signature Scheme ([16] and [2]): An (undirected) transitive signature scheme, which utilizes a standard signature scheme $SS = (SKeyGen, SSign, SVerify)$, is a tuple of six algorithms $TS = (KeyGen, NCert, ESign, VCert, EVerify, Comp)$ such that:

- The algorithm *KeyGen* is the probabilistic *key generation* algorithm. It takes as input 1^k, where k is the security parameter, and calculates a pair (PK, SK) consisting of a *master* public key and a matching secret key. It also calculates a matching key pair (SPK, SSK) for the standard signature using *SKeyGen* algorithm. At last, it outputs the pair $((PK, SK) , (SPK, SSK))$.
- The *node certification* algorithm *NCert*, can be stateful, randomized, or both. It takes as input the master secret key *SK*, the standard signature secret key *SSK*, and a node number n, and produces a node name i and a pair (pk_i, sk_i) consisting of a public key (label) and a matching secret key (label) for node i. It then produces a

signature σ_i using *SSign* algorithm with signing key *SSK* and some message related to i and pk_i. The algorithm finally outputs $(i, (pk_i, sk_i), \sigma_i)$.

- The *edge signing* algorithm *ESign*, which could be stateful, randomized, or both, takes as input the master secret key *SK*, two nodes i and j, and the corresponding secret keys of the nodes sk_i and sk_j, and either computes a value called an *original* signature on edge ij, namely σ_{ij}, or fails.
- The deterministic *certificate verification* algorithm *VCert*, takes as input the standard public key *SPK* and the node certificate (i, pk_i, σ_i), checks the validity of the node certificate using algorithm *SVerify* with verification key *SPK* and returns the validity as a Boolean value.
- The deterministic *edge verification* algorithm *EVerify*, taking the master public key *PK*, two node public keys pk_i and pk_j, and a candidate signature σ_{ij} as input, returns a Boolean value representing the validity of σ_{ij} as a signature of edge ij relative to the public keys.
- The deterministic *composition* algorithm *Comp*, given as input the master public key *PK*, three node public keys pk_i, pk_j and pk_k, and two signatures σ_{ij} and σ_{jk} on nodes ij and jk, either returns a value σ_{ik} as a signature on node ik or fails.

This definition resembles the definitions of [15] and [2] in structure. We also used the ideas of [16]. The paradigm of node certification is used for the public node keys to be brought to others authenticated. By verifying the node certificate to be valid, one can obtain an authenticated message from the original signer saying: "Public key of node i is pk_i." To prove that an edge is in the signed graph, one has to present an *integrated signature* of an edge containing the certificates of its endpoints plus the edge signature and verification of the integrated signature involves verifying both parts. This issue seems to be uncovered in the definition of [16]; therefore we use a mixed definition of [2] and [16], which follows.

It is worth to mention the modification we made in previous definitions. Here, in our definition, we omitted the input i to the node certification algorithm and let the algorithm to choose the node name itself. We later show that by choosing i randomly, the relaxation in the security requirement for *SS* can be achieved. Besides, we know that the algorithm *NCert* is run when the original signer wants to add a new node to the signed graph or wants to recall a certificate. In such a time, there is no difference for the signer what the node name will be. Therefore our modification does not deprive the signer of an important capability, while providing the advantage of short node signatures.

Correctness of Transitive Signature Schemes [2]: As any signature scheme, an original signature is required to be valid with respect to its relative public keys. Furthermore the composition algorithm is required to always produce a valid signature, given as input two valid signatures, either original or composed ones.

Privacy of Transitive Signature Schemes: As stated before, to provide privacy, a valid composed signature must be indistinguishable from a valid original signature on the same edge, which could have been produced by the master signer [15]. This allows using composed signatures as the original ones. In transitive signature schemes whose *ESign* algorithm is deterministic, being indistinguishable reduces to being the same. This means that the composed signature must be equal to the original signature which could have been produced by the master signer.

Security of Transitive Signature Schemes ([15], [2], and [16]): A transitive signature scheme *TS* is called *transitively unforgeable under adaptive chosen message attack* if the advantage in attacking the scheme is negligible for any adversary *F* whose running time is polynomial in the security parameter *k*. The advantage of the best adversary is also known as *insecurity function* and is denoted by

$$InSec_{TS}^{tu-acma}(k).$$

The advantage of *F* in its attack on *TS* is the function defined as

$$Adv_{TS,F}^{tu-acma}(k) = \Pr\left[Exp_{TS,F}^{tu-acma}(k) = 1\right],$$

where the probability is taken over all the random choices made in experiment.

Associated to every transitive signature *TS = (KeyGen, NCert, ESign, VCert, Verify, Comp)*, adversary *F*, and security parameter *k* is an experiment, denoted

$$Exp_{TS,F}^{tu-acma}(k),$$

that returns 1 if and only if *F* is successful in its attack on the scheme and 0 otherwise. The experiment begins by running *KeyGen* on input 1^k to get key pair *(PK, SK)*. It then runs *F*, providing this adversary with input *PK* and oracle access to the function $ESign(SK, \cdot, \cdot, sk_i, sk_j)$, i.e. it can ask to add any new edge *ij* of its choice to the graph and have the signature σ_{ij} on it. Besides, *F* has a certain kind of limited oracle access to function $NCert(SK, SSK, \cdot)$ such that it cannot have access to the part of algorithm output representing the node secret key, i.e. it can query the oracle on any node number *n* and have only the node name *i*, the public node key pk_i and the node signature σ_i. In other words, the adversary can ask to add any new node to the graph and have the certificate (i, pk_i, σ_i) on it. Eventually, *F* will output i', $pk'_{i'}$, j', $pk'_{j'}$ and values $\sigma'_{i'}$, $\sigma'_{j'}$, and $\sigma'_{i'j'}$. Let *E* be the set of all edges such that *F* made oracle query to $ESign(SK, \cdot, \cdot, sk_i, sk_j)$, and let *V* be the set of all nodes which are endpoints of edges in *E*. We say that *F* wins if

$$VCert(i', pk'_{i'}, \sigma'_{i'}) = \text{true},$$
$$VCert(i', pk'_{i'}, \sigma'_{i'}) = \text{true},$$
$$EVerify(PK, pk'_{i'}, pk'_{j'}, \sigma'_{i'j'}) = \text{true},$$

and yet the edge $i'j'$ is not in the transitive closure of the graph $G = (V, E)$. The experiment returns 1 if *F* wins and 0 otherwise.

2.3 GDH Groups, Bilinear Maps and Bilinear Group Pairs

Let us first review the formal definitions and notations of the well-known Diffie-Hellman problems. Resembling [7], we use the following notations: G_1 and G_2 are two (multiplicative) cyclic groups of prime order *p*, with respective generators of g_1 and g_2. By ψ we mean an isomorphism from G_2 to G_1 with $\psi(g_2) = g_1$. The definitions are simplified versions of those in [7].

Computational co-Diffie-Hellman (co-CDH) on (G_1, G_2): Given g_2, g_2^a in G_2 and h in G_1, compute h^a in G_1.

Decision co-Diffie-Hellman (co-DDH) on (G_1, G_2): Given g_2, g_2^a in G_2 and h, h^b in G_1, decide whether $a = b$ or not. When $a = b$ we call (g_2, g_2^a, h, h^b) a (valid) co-Diffie-Hellman *tuple*.

When $G_1 = G_2$ these problems reduce to standard CDH and DDH problems. The advantage of an algorithm A in solving the co-CDH problem on (G_1, G_2) is defined as the probability that A solves the problem correctly given a random instance of the problem specified by a randomly chosen pair for a and h. The probability is taken over the coin tosses of A and the random choices of a and h. We show this probability by

$$Adv_A^{co-CDH}.$$

The maximum advantage among all polynomial time algorithms solving co-CDH problem is also called *insecurity* of co-CDH and is denoted by $InSec^{co\text{-}CDH}$. Co-CDH is called hard if and only if its insecurity is negligible.

Co-GDH Group Pair: The group pair (G_1, G_2) is called a Gap co-Diffie-Hellman group pair if it satisfies the following properties:

1. The group operation on both groups and the map ψ can be computed in one time unit.
2. The co-DDH problem on (G_1, G_2) can be solved in one time unit.
3. The co-CDH problem is hard on (G_1, G_2).

In the above definition, if $G_1 = G_2$, then G_1 is said to be a GDH group.

Bilinear Maps: A function $e: G_1 \times G_2 \rightarrow G_T$, where $|G_1| = |G_2| = |G_T|$, is bilinear if it satisfies the two properties:

1. For every u in G_1 and every v in G_2, and all integers a and b, $e(u^a, v^b) = e(u,v)^{ab}$.
2. e is non-degenerate, i.e. $e(g_1,g_2) \neq 1$.

Bilinear Group Pair: Two order-p groups (G_1, G_2) are called a bilinear group pair if they satisfy the following properties:

1. The group operation on both groups and the map ψ can be computed in one time unit.
2. A group G_T of order p and a bilinear map $e: G_1 \times G_2 \rightarrow G_T$ exist and e is computable in one time unit.
3. The co-CDH is hard on (G_1, G_2).

Joux and Nguyen [13] showed that an efficiently computable bilinear map e provides an algorithm for solving the co-DDH problem as follows:

$$a = b \bmod p \quad \Leftrightarrow \quad e\left(h, g_2^a\right) = e\left(h^b, g_2\right).$$

As a consequence, if two groups are a bilinear group pair, then they are also a co-GDH group pair. The converse is probably not true [7].

3 A Transitive Signature on Bilinear Group Pairs (BGPTS)

We present a transitive signature scheme, based on the BLS signature ideas, that works in bilinear group pairs. We assume that we have a standard signature scheme $SS = (SKeyGen, SSign, SVerify)$, whose message space is the set of all strings made by concatenating a member of the definite set of all node names with a member of the group G_1. We construct the transitive signature scheme, using this standard signature and a bilinear group pair (G_1, G_2). At last, we prove our transitive signature scheme transitively secure under chosen message attack.

Let the bilinear group pair generator G_{BGP} as a randomized polynomial time algorithm that on input 1^k, where k is the security parameter, generates a bilinear group pair (G_1, G_2), where $|G_1| = |G_2| = p(k)$ and the insecurity of the co-CDH problem on (G_1, G_2) is the amount $InSec^{co\text{-}CDH}(k)$. It is obvious that for the group pair to be a bilinear one, this function must decrease faster than any polynomial in k. The formal description of the transitive signature *BGPTS* follows:

- The key generation algorithm *KeyGen*, takes an input 1^k, runs G_{BGP} on this value, and obtains the bilinear group pair (G_1, G_2). It then picks a random member of $Z_{p(k)}$, namely SK, and computes $PK \leftarrow g_2^{SK}$. The algorithm also runs *SKeyGen* on input 1^k to obtain a key pair (SPK, SSK) for the standard signature. At last, the algorithm outputs the group pair (G_1, G_2), the master key pair (PK, SK), and the standard key pair (SPK, SSK).
- The node certifications algorithm *NCert* maintains state *NodeList*, where *NodeList* is a list containing the set of all so-far queried nodes, their secret and public node keys (labels), and their signatures. On input SK, SSK, and node number n, the algorithm checks if n is on the list. If so, it outputs the corresponding node name, secret node key, and public node key and the corresponding node signature from *NodeList*. Otherwise, it picks a random member of the set of all node names, namely i, and also picks a random member of G_1, namely sk_i. The algorithm then computes $pk_i \leftarrow sk_i^{SK}$ and runs *SSign* on inputs SSK and the message $i \,||\, pk_i$ and obtains σ_i. The algorithm then outputs the node name i and the pair (pk_i, sk_i) as the matching public and secret node keys followed by the node signature σ_i.
- The edge signing algorithm *Esign* takes as input sk_i and sk_j and simply outputs
$$\sigma_{ij} \leftarrow sk_i / sk_j \ .$$
- The certificate verification algorithm *VCert*, on input (i, pk_i, σ_i), runs *SVerify* and outputs $SVerify(SPK, i \,||\, pk_i\,, \sigma_i)$.
- The verification algorithm *EVerify*, takes as input PK, pk_i, pk_j, and a candidate signature σ_{ij} and verifies that
$$\left(g_2, PK, \sigma_{ij}, pk_i / pk_j\right)$$
is a (valid) co-Diffie-Hellman tuple. If so, it returns true; if not, returns false.
- The composition algorithm *Comp*, given as input σ_{ij} and σ_{jk}, computes and outputs the value $\sigma_{ik} \leftarrow \sigma_{ij} \cdot \sigma_{jk}$.

Note that, sometimes, it is needed to have σ_{ij}, and sometimes σ_{ji} to compose signatures correctly. Yet since they are inverses of each other and inversion in the multiplicative group G_1 can be made in polynomial time, the transformations $\sigma_{ij} \leftrightarrow \sigma_{ji}$

are both feasible. Therefore, these transformations are omitted from the descriptions of the algorithms.

Eliminating State: As proposed in [2], state *NodeList* can be eliminated through using a pseudorandom function. The master secret key could include a key K to specify an instance F_K from a pseudorandom function family F. Then the signer does not need to save anything. He/She just uses $F_K(i)$ for all the coins needed for the node name and the node keys.

Correctness: It is easy to see that any original signature σ_{ij} produced by the master signer leads to a true output by the verification algorithm. On the other hand, any composed signature of two valid (original or composed) signatures leads to the same output in verification. The proof is up to the reader!

Privacy: The *ESign* algorithm of BGPTS scheme is deterministic. As the composed signature in BGPTS scheme is the same as the signature that could have been produced by the master signer, the privacy property requirement for the signature scheme is met.

Security: As an intuition, it is worth to see that a verifier of a signature σ_{ij} faces a co-DDH problem instance in a bilinear group pair, which is assumed to be easy. On the other side is a forger of the signature. If it wants to forge a signature on an edge, one of whose endpoints is not certified so far, it faces the problem of forging a valid signature for the standard signature scheme, which is infeasible by the assumption of the standard signature's security. Otherwise, it faces the problem of forging a signature on an edge whose endpoints are both certified by the master signer. In this case, it knows three arguments out of four of a tuple, namely g_2, PK, and pk_i / pk_j, and it wants to compute σ_{ij} so as the tuple be a valid Diffie-Hellman tuple. This is, obviously, a co-CDH problem instance in a bilinear group pair, which is assumed to be infeasible. The following theorem states that the BGPTS is transitively secure assuming that underlying primitives are flawless. Our method for proving the security of BGPTS is partly similar to Coron's method for proving that of FDH scheme in [9].

Security Theorem: The BGPTS scheme described above is transitively unforgeable under adaptive chosen message attack, assuming that G_{BGP} produces a bilinear group pair and standard signature scheme *SS* is unforgeable under known message attack. More precisely, if the most advantageous adversary asks a maximum of q' queries from the node certification oracle and a maximum of q queries from the edge signing oracle in attacking BGPTS, we have

$$\exp(1)\cdot q \cdot InSec^{co-CDH}(k) + InSec_{SS}^{uf-kma}(k) \geq InSec_{BGPTS}^{tu-acma}(k),$$

where k is the security parameter of the transitive signature input to the key generation algorithm. Furthermore, if we denote the running time of *SSign* and *Esign* algorithms by t_{SSign} and t_{ESign} and that of the best adversaries attacking BGPTS, SS, and co-CDH by t_{BGPTS}, t_{SS}, and $t_{co\text{-}CDH}$, we have

$$t_{BGPTS}(k) \geq \max\Big\{t_{co-CDH}(k) - (q'+q)\cdot O\big(p^3(k)\big) - q'\cdot t_{SSign}(k),\\ t_{SS}(k) - q'\cdot O\big(p^3(k)\big) - q\cdot t_{ESign}(k)\Big\}.$$

Proof Sketch: We will prove the security by reduction as follows. Given any polynomial time forger F for BGPTS asking at most q queries from the edge signing

oracle, We will show how to use F to construct two algorithms: An algorithm A to solve the co-CDH problem in (G_1, G_2) and a forger F' breaking the underlying standard signature SS through a known message attack, such that

$$\alpha(q)\cdot Adv_A^{co-CDH}(k)+Adv_{SS,F'}^{uf-kma}(k)=Adv_{BGPTS,F}^{tu-acma}(k),$$

$$\text{where}\quad \alpha(q)=\frac{1}{\left(1-\frac{1}{q+1}\right)^{q+1}}\cdot q.$$

We must show that $\alpha(q)$ is bounded by a polynomial in q, and hence the coefficient $\alpha(q)$ grows polynomially in q, however the advantage of A in solving co-CDH decreases faster than any polynomial in k. This means that controlling k, we can keep the advantage of the adversary attacking BGPTS sufficiently low.

Note that we have

$$\lim_{q\to\infty}\frac{1}{\left(1-\frac{1}{q+1}\right)^{q+1}}=e,$$

therefore $\alpha(q)$ can be bounded linearly in q, i.e. $\alpha(q) = O(q)$.

Since there could be more efficient ways to construct algorithms A and F', the equation

$$\exp(1)\cdot q\cdot InSec^{co-CDH}(k)+InSec_{SS}^{uf-kma}(k)\geq InSec_{BGPTS}^{tu-acma}(k)$$

is proven for large q.

The full description of how the two algorithms A and F' are constructed comes in the full proof of the security theorem in the appendix. We just mention that in the proof, techniques of proving security in [2], [7], and [9] are mixed together.

As in [2], we show that signatures for BGPTS can be forged in only two ways: either there is the forgery that "recycles node certificates from previously issued signatures", or there is the forgery that "includes at least one new node certificate". We will show that the former type of forgery leads us to solve a certain co-CDH problem with a certain probability of success, while a forgery of the latter type can be easily transformed to an attack on SS: the new node certificate is a valid forgery for SS, as it contains a standard node signature that was not produced by the original signer before.

In simulating *NCert* algorithm, when algorithm A is answering oracle queries made by F, we use the technique of [7]. We simply embed the h argument of our co-CDH instance in some simulated node public keys, while choosing other simulated node public keys randomly. We call the former type of nodes *h-node* and the latter *non-h-node*. Then, similar to [7] again, A can answer F's *ESign* oracle queries only when the edge endpoints are nodes of a type, and succeeds in solving the co-CDH instance it has only when the edge endpoints of the forgey provided by F are nodes of different types.

To get the best success probability in our attack, we use the technique of [9]. We just embed the h argument in simulated node public keys with a certain probability p_0, and choose other simulated node public keys randomly with probability $1 - p_0$. This

leads us to maximize A's success probability and hence optimize the security by carefully selecting p_0 with respect to q. The function $\alpha(q)$ is originally an optimized version of a function of two arguments q and p_0 minimized with respect to p_0.

As a time domain analysis, since both t_{SS} and $t_{co\text{-}CDH}$ grow faster than any polynomial in k, assuming the standard signature secure and the co-CDH problem hard, the time complexity for the best adversary attacking BGPTS also grows faster than any polynomial in k, for the reason that other subtractive terms in the time complexity equation above are polynomial in k.

We refer the reader to the appendix of this paper for the full description of the proof.

Eliminating Node Certification via Hashing: As stated comprehensively in [2], node certification brings us the disadvantages of "increasing the signature size as well as the computational cost for signing and verifying". Resembling [2], we can eliminate node certificates by specifying the public key of the node i via the output of a hash function by

$$i \,\|\, pk_i \leftarrow H(i)$$

and then setting

$$sk_i \leftarrow pk_i^{1/SK} .$$

This provides an "implicit authentication" [2] of node public keys, i.e. there is no need for the original signer to certify nodes anymore. As a consequence, the node certification algorithm collapses to node key generation and the certificate verification algorithm will no more exist. This means that there will be no further need for the standard signature to sign node public keys and verifying them. Fully-described changes in BGPTS are routine and similar to [2] and therefore are omitted here. It is just worth to state that the security of the new scheme relies on the hardness of the co-CDH problem in bilinear group pairs, in the so called *random oracle model (ROM)*. In this model, hash functions are viewed as random functions (See [4] and [5] for further on ROM.).

4 Conclusions

We have constructed a short transitive signature scheme from bilinear maps whose security is proven under reasonable assumptions, such as hardness of the computational Diffie-Hellman problem and existence of secure standard signatures. Shortness of an edge signature is due to the fact of using bilinear group pairs with small representations and that of a node signature is due to the fact of using signatures which are required to be secure only under known message attack. These two, eventually, yield to a very short signature on the whole graph, which is very probable to have a big size in everyday applications. This fact, finally, results in a lower amount of communication traffic load.

Acknowledgement

We firstly thank the anonymous reviewers of the SCN'04 conference for their precious comments. In fact, the idea of eliminating node certification via hashing was

a simultaneous idea of a reviewer and the authors. Afterwards, we would like to thank Taraneh Eghlidos for the support and encouragement she gave and Iman Mosavat for helpful discussions, as well. We also thank Mofid Nakhaei, Sara Zaeemdar, and Ebrahim Hedayati for their carefully reading the text and proposing editions.

References

1. P. Barreto. “The Pairing-Based Crypto Lounge” Web Page. *http://planeta.terra.com.br/informatica/paulobarreto/pblounge.html*.
2. M. Bellare and G. Neven. “Transitive Signatures Based on Factoring and RSA”. Asiacrypt 2002. LNCS Vol. 2501. Springer-Verlag. 2002.
3. M. Bellare and G. Neven. “Transitive Signatures: New Schemes and Proofs”. Cryptology ePrint Archive: Report 2004/215. *http://eprint.iacr.org/2004/215/*. Also at *http://www.cse.ucsd.edu/users/mihir/*. (Full version of [2]).
4. M. Bellare and P. Rogaway. “Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols”. The First Annual Conference on Computer and Communications Security. ACM 93. 1993.
   Full Paper: *http://www-cse.ucsd.edu/users/mihir*.
5. M. Bellare and P. Rogaway. “The Exact Security of Digital Signatures – How to Sign with RSA and Rabin”. Eurocrypt 96. LNCS Vol. 1070. Springer-Verlag. 1996.
   Full Paper: *http://www-cse.ucsd.edu/users/mihir*.
6. D. Boneh, M. Franklin. “Identity Based Encryption from the Weil Pairing”. SIAM Journal of Computing Vol. 32 No. 3 pp. 586-615. Extended Abstract in Crypto 2001.
   Full Paper: *http://crypto.stanford.edu/~dabo/pubs.html*.
7. D. Boneh, B. Lynn, and H. Shacham. “Short Signatures from the Weil Pairing”. Asiacrypt 2001. LNCS Vol. 2248. Springer-Verlag. 2001.
   Revised Full Paper: *http://crypto.stanford.edu/~dabo/pubs.html*.
8. D. Boneh, I. Mironov, and V. Shoup. “A Secure Signature Scheme from Bilinear Maps”. CT-RSA 2003. LNCS Vol. 2612. Springer-Verlag. 2003.
   Full Paper: *http://crypto.stanford.edu/~dabo/pubs.html*.
9. J. S. Coron. “On the Exact Security of Full Domain Hash”. Crypto 2000. LNCS Vol. 1880. Springer-Verlag. 2000.
   http://www.gemplus.com/smart/r_d/publications/pdf/Cor00fdh.pdf.
10. S. Goldwasser, S. Micali, and R. Rivest. “A digital signature scheme secure against adaptive chosen-message attacks”. SIAM Journal of Computing Vol. 17 No. 2 April 1988 pp. 281-308.
    http://theory.lcs.mit.edu/~rivest/publications.html.
11. A. Hevia and D. Micciancio. “The Provable Security of Graph-Based One-Time Signatures and Extensions to Algebraic Signature Schemes”. Asiacrypt 2002. LNCS Vol. 2501. Springer-Verlag. 2002.
    http://www.cs.ucsd.edu/~ahevia/publications/hm02.pdf.
12. R. Johnson, D. Molnar, D. Song, and D. Wagner. “Homomorphic Signature Schemes”. CT-RSA 2002. LNCS Vol. 2271. Springer-Verlag.
    http://citeseer.nj.nec.com/460118.html.
13. A. Joux and K. Nguyen. “Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups”. Journal of Cryptology Vol. 16 No. 4 pp. 239-247 Sep. 2003. A previous version also available online: “Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Protocols”. Cryptology ePrint Archive. Report 2001/003. *http://eprint.iacr.org/2001/003*.

14. H. Lipma0a. "Pairing-based Cryptography" Web Page on Cryptology Pointers. *http://www.tcs.hut.fi/~helger/crypto/link/public/pairing/*.
15. S. Micali and R. Rivest. "Transitive Signature Schemes". CT-RSA 2002. LNCS Vol. 2271. Springer-Verlag. *http://citeseer.nj.nec.com/micali02transitive.html*.
16. D. Molnar. "Homomorphic Signature Schemes". BA Thesis, Computer Science Dept. Harvard College. Cambridge. Massachusetts. Michael Rabin adv. 2003. *http://www.cs.berkeley.edu/~dmolnar/papers/papers.html*.
17. G. Neven. "Provably secure identity-based identification schemes and transitive signatures". Ph.D. thesis. Katholieke Universiteit Leuven, Belgium. May 2004. *http://www.cs.kuleuven.ac.be/~gregory/papers/phd.html.*
18. R. Rivest. "Two New Signature Schemes". Slides from Talk Given at Cambridge University. 2000. *http://www.cl.cam.ac.uk/Research/Security/seminars/2000/rivest-tss.pdf*.
19. Z. Sujing. "Transitive Signatures Based on Non-adaptive Standard Signatures". Cryptography ePrint Archive. Report 2004/044. *http://eprint.iacr.org*.

Appendix: Proof of Security Theorem

Suppose we are given a feasible forger F for BGPTS. We will show how to use F to construct an algorithm A to solve the co-CDH problem in (G_1, G_2) and a forger F' breaking the underlying standard signature SS through a known message attack, such that

$$\alpha(q)\cdot Adv_A^{co-CDH}(k)+Adv_{SS,F'}^{uf-kma}(k)=Adv_{BGPTS,F}^{tu-acma}(k),$$

where q is the number of *ESign* queries F makes during its attack on BGPTS and

$$\alpha(q)=\frac{1}{\left(1-\frac{1}{q+1}\right)^{q+1}}\cdot q.$$

Note that for every q we have

$$2q<\alpha(q)\le 4q.$$

Hence, the forger's advantage grows linearly in q, but it descends faster than any polynomial in k. Therefore is proven the security of BGPTS.

Algorithm A performs as follows: given g_2, $u = g_2^a$ in G_2 and h in G_1 as input, it computes h^a in G_1. It maintains state *State* in which it saves the data it will need later through the algorithm run, such as queries made by F and A's corresponding answers. Using this state A can simply answer repeated queries by repeated answers and just calculate answers to *new* queries. It first generates a fresh key pair *(SPK, SSK)* for SS using the algorithm *SKeyGen*. Then It computes $v = \psi(u)$, which will be used later. Note that since ψ is assumed to be an isomorphism, we have:

$$v=\psi(u)=\psi\left(g_2^a\right)=\psi(g_2)^a=g_1^a.$$

Then the algorithm A runs F on input $PK = u \cdot g_2^r = g_2^{a+r}$, where r is chosen randomly from Z_p by algorithm A. Now F will start its *ESign(SK, ·, ·, sk_i, sk_j)* and *NCert(SK, SSK, ·)* oracle queries. As A does not know SK it cannot answer the queries

by simply running the *ESign* and *NCert* algorithms. Therefore it will simulate these two algorithms as follows.

On an *NCert(SK, SSK, ·)* query for adding a new node *n* and certifying it, algorithm *A* first chooses a random node name *i* from the set of all node names and a random b_i from Z_p. Then it produces a random coin $c_i \in \{0,1\}$, where $c_i = 0$ with probability p_0 and $c_i = 1$ with probability $1 - p_0$. The value p_0 is a fixed probability chosen to get a better reduction (idea from [9]) and will be determined later in this paper. If $c_i = 0$ it sets $pk_i \leftarrow h \cdot g_1^{b_i}$. Otherwise it sets $pk_i \leftarrow g_1^{b_i}$. At last it answers to the query by outputting three values representing the name *i*, the public key pk_i and the certification signature $\sigma_i = SSign(SSK, i \,||\, pk_i)$ of the corresponding node. It also saves the values *i*, b_i, and c_i for the node *n* by updating its state *State*. Note that since b_i is random, both $h \cdot g_1^{b_i}$ and $g_1^{b_i}$ are randomly distributed over G_1 and are indistinguishable for the algorithm *F* from each other and from a real public node key which could have been produced by a real signer. Therefore the simulation is flawless and also *F* has no idea what c_i could be for a node *i*.

Before we describe how to simulate answers to *ESign* queries, we introduce a notation we will use in the description. We simply call a node *i* an "h-node" if $c_i = 0$ and call it a "non-h-node" otherwise. We also assume that when the algorithm *F* queries its *ESign* oracle on the edge *ij* it has already queried its *NCert* oracle on nodes named *i* and *j* for their certificate. This assumption can be justified since any node name *i* and the corresponding node keys are chosen independently. Moreover the edge signature is also independent of any single node key. As a result, if at least one of the nodes is not queried before for its certificate, the answer to the *ESign* query will be just a random value independent of other things *F* knows and will be of no use for it.

On an *ESign(SK, ·, ·, sk_i, sk_j)* query for signing the edge *ij* of the graph, Algorithm *A* looks in *State* to recognize one the two possible cases bellow:

1. If *i* and *j* are nodes of a type, i.e. both are non-h-nodes or both are h-nodes, then *A* simply answers the query as $\sigma_{ij} = \left(v \cdot g_1^{r}\right)^{b_i - b_j}$.

2. If one of *i* and *j* is an h-node and the other one is a non-h-node, then *A* reports failure and terminates.

Note that in the first case we have:

$$\sigma_{ij} = \left(v \cdot g_1^{r}\right)^{b_i - b_j} = \left(g_1^{a} \cdot g_1^{r}\right)^{b_i - b_j} = \left(g_1^{b_i - b_j}\right)^{a+r} \text{ and } pk_i / pk_j = g_1^{b_i - b_j}.$$

Hence the tuple $(g_2, PK = g_2^{a+r}, pk_i / pk_j, \sigma_{ij})$ is a valid co-Diffie-Hellman tuple and σ_{ij} is a valid signature for edge *ij*. Therefore the simulation works properly.

Finally *F* will output a forgery including $i', pk'_{i'}, j', pk'_{j'}$ and values $\sigma'_{i'}$, $\sigma'_{j'}$, and $\sigma'_{i'j'}$. *A* will use this output to solve the co-GDH problem, assuming that *F* manages to win, i.e. manages to forge valid signatures. More precisely, let *E* is the set of all edges such that *F* made an *ESign* oracle query on, and let *V* be the set of all nodes which are endpoints of edges in *E*. Winning for *F* means that:

$$\Sigma_1: VCert(i', pk'_{i'}, \sigma'_{i'}) = \text{true},$$
$$\Sigma_2: VCert(i', pk'_{i'}, \sigma'_{i'}) = \text{true},$$
$$\Sigma_3: EVerify(PK, pk'_{i'}, pk'_{j'}, \sigma'_{i'j'}) = \text{true},$$

and yet Σ_4: the edge $i'j'$ is not in the transitive closure of the graph $G = (V, E)$.

We call these statements Σ_1, Σ_2, Σ_3, and Σ_4, respectively. The statement Σ_3 specifically means that $(g_2, PK = g_2^{a+r}, pk'_{i'} / pk'_{j'}, \sigma'_{i'j'})$ is a valid co-Diffie-Hellman tuple, i.e.

$$\sigma'_{i'j'} = \left(pk'_{i'} / pk'_{j'}\right)^{a+r}.$$

Algorithm A now checks that if the node public keys returned by F match those it produced itself or not, i.e. it checks the statements Σ_5 introduced bellow:

$$\Sigma_5: pk'_{i'} = pk_{i'} \text{ and } pk'_{j'} = pk_{j'}.$$

If Σ_5 is not true Algorithm A reports failure and terminates. Otherwise, it checks *State* to find out if i' and j' are nodes of a type. If so, A reports failure and terminates. Otherwise, there are two possibilities:

1. i' is an h-node and j' is a non-h-node. In this case, as we have:

$$\sigma'_{i'j'} = \left(h \cdot g_1^{b_{i'}} / g_1^{b_{j'}}\right)^{a+r}.$$

So A simply computes and outputs h^a as:

$$h^a = \sigma'_{i'j'} / \left(h^r \cdot g_1^{r\left(b_{i'} - b_{j'}\right)} \cdot v^b\right).$$

2. i' is a non-h-node and j' is an h-node. In this case, as we have:

$$\sigma'_{i'j'} = \left(g_1^{b_{i'}} / h \cdot g_1^{b_{j'}}\right)^{a+r}.$$

So A simply computes and outputs h^a as:

$$h^a = \left(g_1^{r\left(b_{i'} - b_{j'}\right)} \cdot v^b\right) / \left(h^r \cdot \sigma'_{i'j'}\right).$$

For calculating the success probability of the algorithm A, we observe that it succeeds whenever it does not report failure. First, if F asks q queries from its oracle *ESign*, algorithm A can answer all the q queries with probability $[p_0^2 + (1- p_0)^2]^q$. This is true for the reason that, in each query, two nodes i and j are both h-nodes with probability p_0^2 and are both non-h-nodes with probability $(1- p_0)^2$. Secondly, F will succeed in case that the nodes i' and j' are of two different types. The probability that this occurs is $2\ p_0\ (1- p_0)$. In these calculations we used the fact that simulation is correct, i.e. h-nodes and non-h-nodes are indistinguishable.

Finally, by defining $\beta(q, p_0) = 2\ p_0\ (1- p_0)\ [p_0^2 + (1- p_0)^2]^q$, we can calculate the advantage of algorithm A with respect to the advantage of algorithm F as follows:

$$\begin{aligned} Adv_A^{co-CDH}(k) &= \beta(q, p_0) \cdot \Pr\left[\bigwedge_{t=1}^{5} \Sigma_t\right] \\ &= \beta(q, p_0) \cdot \Pr\left[\Sigma_5 \mid \bigwedge_{t=1}^{4} \Sigma_t\right] \cdot \Pr\left[\bigwedge_{t=1}^{4} \Sigma_t\right] \\ &= \beta(q, p_0) \cdot \Pr\left[\Sigma_5 \mid \bigwedge_{t=1}^{4} \Sigma_t\right] \cdot Adv_{BGPTS,F}^{tu-acma}(k). \end{aligned}$$

Algorithm F' is given SPK as input and a list of random messages and corresponding signatures. It will perform a known message attack on SS using F as a subroutine. Its goal is to eventually output a message-signature pair, where the signature is a valid signature for the message with respect to SPK and yet the message was not on the message-signature list provided for the adversary.

The algorithm F' first runs the algorithm $KeyGen$ of the transitive signature to obtain a pair of keys (PK, SK). It then runs F on input PK and answers the queries of F to the oracles $ESign(SK, \cdot, \cdot, sk_i, sk_j)$ and $NCert(SK, SSK, \cdot)$ as follows:

On an $NCert(SK, SSK, \cdot)$ query n, F' first chooses the n-th entry in the message-signature pair list. It then parses the corresponding message as $i \| pk_i$. Afterwards, it computes sk_i as

$$sk_i \leftarrow pk_i^{1/SK}.$$

It also sets the corresponding signature as σ_i. Note that as messages in the message-signature pair list are randomly chosen, both i and pk_i are random and hence is sk_i. Therefore, the simulation works correctly. Moreover, the probability that the node name is repeated in the list is a small *constant* value and we do not take it into account. The reason is that the size of the set of all node names is constant and independent of the security parameter k.

On an $ESign(SK, \cdot, \cdot, sk_i, sk_j)$ query on edge ij, assuming that F has previously queried the two certificates on both nodes, F' looks i and j up in the message-signature pair list and finds the corresponding public and secret node keys. Now, since F' knows SK, sk_i, and sk_j, it simply runs the $ESign$ algorithm of the transitive signature scheme and provides the output σ_{ij} as the answer to the query.

Eventually, F outputs a forgery including i', $pk'_{i'}$, j', $pk'_{j'}$ and values $\sigma'_{i'}$, $\sigma'_{j'}$, and $\sigma'_{i'j'}$. Assuming that F wins, i.e. the statements Σ_1, Σ_2, Σ_3, and Σ_4 are all true, F' checks that if the node public keys returned by F match those it produced itself or not, i.e. it checks the statements Σ_5. If Σ_5 is true Algorithm F' reports failure and terminates. Otherwise, at least one of $pk'_{i'}$, $pk'_{j'}$ was not certificated by F' before. In other words, at least one of $i' \| pk'_{i'}$ and $j' \| pk'_{j'}$ is not a message in the message-signature pair list. Therefore, at least one of the signatures $\sigma'_{i'}$ and $\sigma'_{j'}$ must be a forgery, and is obviously a valid one because Σ_1 and Σ_2 are both true. Hence, all F' has to do is to test whether the string $i' \| pk'_{i'}$ is not a message in the list and output $(i' \| pk'_{i'}, \sigma'_{i'})$ as a new message-signature pair representing forgery if so, or output $(j' \| pk'_{j'}, \sigma'_{j'})$ otherwise.

Algorithm F' succeeds whenever all the statements Σ_1, Σ_2, Σ_3, and Σ_4 are true, but Σ_5 is not true. This fact yields to the calculation bellow:

$$\begin{aligned}
Adv_{SS,F'}^{uf-kma}(k) &= \Pr\left[\left(\bigwedge_{t=1}^{4}\Sigma_t\right)\wedge\overline{\Sigma_5}\right] \\
&= \Pr\left[\overline{\Sigma_5} \mid \bigwedge_{t=1}^{4}\Sigma_t\right]\cdot\Pr\left[\bigwedge_{t=1}^{4}\Sigma_t\right] \\
&= \Pr\left[\overline{\Sigma_5} \mid \bigwedge_{t=1}^{4}\Sigma_t\right]\cdot Adv_{BGPTS,F}^{tu-cma}(k) \\
&= \left(1-\Pr\left[\Sigma_5 \mid \bigwedge_{t=1}^{4}\Sigma_t\right]\right)\cdot Adv_{BGPTS,F}^{tu-acma}(k).
\end{aligned}$$

By eliminating the repeating term in the two calculations we did for the success probability of A and F', we will simply reach the equation:

$$\frac{1}{\beta(q, p_0)} \cdot Adv_A^{co-CDH}(k) + Adv_{SS,F'}^{uf-kma}(k) = Adv_{BGPTS,F}^{tu-acma}(k).$$

To optimize the security, we must maximize the function $\beta(q, p_0)$ by properly selecting p_0 with respect to a given q. Let's rewrite $\beta(q, p_0)$ as:

$$\begin{aligned}\beta(q, p_0) &= 2p_0(1-p_0)\left[p_0^2 + (1-p_0)^2\right]^q \\ &= 2p_0(1-p_0)\left[1 - 2p_0(1-p_0)\right]^q\end{aligned}$$

Defining $p_1 = 2p_0(1-p_0)$ we have

$$\beta(q, p_1) = p_1(1-p_1)^q.$$

The above function is maximized as below:

$$\begin{aligned}&\frac{\partial\beta}{\partial p_1} = 0 \Rightarrow p_1 = \frac{1}{1+q} \\ &\Rightarrow 2p_0(1-p_0) = \frac{1}{1+q} \\ &\Rightarrow p_0 = \frac{1}{2} \pm \sqrt{\frac{1}{4}\frac{q-1}{q+1}}.\end{aligned}$$

Now $\alpha(q)$ is defined and computed as

$$\alpha(q) = \frac{1}{\max\limits_{p_0}\{\beta(q, p_0)\}} = \frac{1}{\left(1 - \frac{1}{q+1}\right)^{q+1}} \cdot q$$

which is the result we were seeking.

The running time of A equals that of F plus one exponentiation and one *SSig* algorithm run for every *NCert* query plus one exponentiation for every *ESign* query, i.e.

$$t_A(k) = t_F(k) + q' \cdot \left(O\left(p^3(k)\right) + t_{SSign}(k)\right) + q \cdot O\left(p^3(k)\right),$$

for that there are at most q' *NCert* queries and at most q *ESign* queries and modular exponentiation time complexity is cubic in group size.

The running time of F' equals that of F plus one exponentiation for every *NCert* query plus one *ESign* algorithm run for every *ESign* query, i.e.

$$t_{F'}(k) = t_F(k) + q' \cdot O\left(p^3(k)\right) + q \cdot t_{ESign}(k),$$

for that there are at most q' *NCert* queries and at most q *ESign* queries.

Since there could be more efficient ways to construct algorithms A and F', the claimed equation for the time complexity is proven.

Errata

The security theorem on page 9 states that the BGPTS scheme is secure under the *co-CDH* assumption. The proof of this theorem is not correct. However, an almost identical scheme, GapTS-1, is shown to be secure under the *one-more CDH* assumption by Bellare and Neven [BN05]. Hence, BGPTS is still secure, albeit under a stronger assumption. Note that BGPTS and GapTS-1 were proposed independently in 2004, respectively in the SCN'04 conference and in [BN04].

[BN02] M. Bellare and G. Neven. "Transitive Signatures Based on Factoring and RSA". Asiacrypt 2002. Y. Zheng ed., LNCS 2501, pp. 397–414, Springer-Verlag, 2002.

[BN04] M. Bellare and G. Neven. "Transitive Signatures: New Schemes and Proofs". Cryptology ePrint Archive, Report 2004/215, 2004. Multiple versions available at `http://eprint.iacr.org/2004/215`

[BN05] M. Bellare and G. Neven. "Transitive Signatures: New Schemes and Proofs". IEEE Transactions on Information Theory, Vol. 51, No. 6, pp. 2133–2151, 2005.