An Authentication Protocol For Mobile Agents Using Bilinear Pairings

Amitabh Saxena, and Ben Soh
Dept. of Computer Science and Computer Engineering
La Trobe University, Bundoora, VIC, Australia 3086

September 1, 2005

Abstract

A mobile agent is a mobile program capable of maintaining its execution states as it migrates between different execution platforms. A key security problem in the mobile agent paradigm is that of trust: How to ensure that the past itinerary (of execution platforms) claimed by the agent is correct. This is necessary in order to establish a reasonable level of trust for the agent before granting execution privileges.

In this paper we describe a protocol using bilinear pairings that enables trust relationships to be formed between agent platforms in an ad-hoc manner without actively involving any trusted third party. This protocol can be used to authenticate agents before granting execution privileges. The main idea behind our approach is the concept of 'one-way' chaining. Our scheme has chosen ciphertext security assuming the hardness of the Bilinear Diffie Hellman Problem (BDHP).

1 Introduction

Mobile agents are agents that can physically travel across networks and perform tasks on machines that provide agent hosting capability. This allows processes to migrate from computer to computer, for processes to split into multiple instances that execute on different machines, and to return to their point of origin. A detailed discussion of mobile agents is beyond the scope of this paper and the reader is referred to [1]. Two foremost security challenges for mobile agents are (a) host protection and (b) agent protection. Our work on mobile agents is focused only on host protection. For work on agent protection the reader is referred to [2, 3, 4, 5].

In contrast to approaches for host protection based on sandbox environments or other forms of code validation, our model aims to validate the itinerary of an agent. Our approach to security is based on a notion of trust which is summarized as follows: If all entities involved with the agent can be authenticated, a level of trust can be established, which can then be used for granting or denying execution privileges. Current solutions for host protection rely on tamper proof hardware, an on line trusted third party or a 'sandbox' model of execution [6, 7, 8]. Our method does not require any such measures. We use the concept of *one-way* signatures to connect arbitrary hosts in a chain of trust, thus enabling ad-hoc trust relationships to be formed.

The concept of one-way signature chaining was proposed in [9] and [10] where the authors constructed authentication protocols for mobile agents using hypothetical cryptographic primitives known as *strong non-commutative associative one-way functions*. The authors also asked if an equivalent protocol can be constructed using any existing cryptographic primitives. In this paper, we answer this question affirmatively and show that the mobile agent authentication protocol presented in [10] can be constructed using bilinear pairings, thus settling their open question.

Although the original concept of signature chaining presented in [10] is based on a standard certificate based Public Key Infrastructure (PKI), it can be shown that their model can be reduced directly to an Identity-Based Public Key Cryptosystem (ID-PKC) or a Certificate-Less Public Key Cryptosystem (CL-PKC) due to certain properties of the one-way function used.[1] In contrast to this, the protocol presented in this paper is based on a standard certificate based PKI and it is not known if a direct reduction to an ID-PKC or a CL-PKC exists.

[1] The reader is referred to [11] for a discussion of an ID-PKC and to [12] for a discussion of a CL-PKC

2 Background

Any entity that runs a mobile agent platform server is called a *host.* We assume that all such hosts are identified by a public directory. Any host that initiated an agent into the system is called the *initiator* of the agent. Agents can migrate autonomously between different host platforms. This act of migration is called *agent transfer.* An *instance* of an agent is a snapshot of its state at any point of execution on some platform. An *itinerary* is the ordered list of hosts visited by an agent.

2.1 Agent partitioning

Using the object oriented paradigm, we assume that any instance of a mobile agent can be split (or partitioned) into a *static* part (consisting of object methods) which is unchanging as the agent hops across platforms and a *dynamic* part (consisting of data and the state information of the interacting objects) that changes at each hop. Depending on the specific implementation, the partitioning schemes can differ. However, in this section we enumerate certain properties relevant in our context.

1. *Unique*: It may be possible that an instance of the agent can be partitioned in more than one ways. A partition scheme is *unique* if all instances of the agent have a unique static and dynamic part.

2. *Identical*: A partition scheme is *identical* if all instances of the agent have at least one common static part.

3. *Mutually authenticating*: We further assume that some static and dynamic parts can be made mutually inseparable. This means that the agent's functionality is available if and only if both the static and dynamic parts correspond to the same agent. Mixing and matching between different agents is not possible. We say that the scheme is *mutually authenticating* if all instances of the agent have at least one mutually inseparable partition.

4. *Ideal*: A partitioning scheme is *ideal* if it is unique, identical and mutually authenticating.

2.2 Authentication Requirements

In this section, we give the high-level authentication requirements for our model. we define the following two requirements:

1. *Initiator authentication*: Is the claimed initiator the same as the real initiator?

2. *Itinerary authentication*: Is the claimed itinerary the same as the real itinerary?

Our requirement for unconditional security is itinerary authentication. It is evident, however, that this will also always involve initiator authentication, since the initiator is the first host in the itinerary. We introduce the concept of *relative authentication* to imply that the first host (the initiator) in an itinerary is unknown. On the other hand, *absolute authentication* implies that the initiator can be authenticated.

2.3 One-way Chaining

Represent the host platforms as points of an acyclic directed graph. As the agent hops, a new arc directed from the receiver to the sender is added to the graph. The edges of such a graph will represent a hop-by-hop path of the agent in the reverse direction from the current host to the initiator. In this notation the statements "a passed the agent to b" and "There is a path of unit length from b to a" are considered equivalent. We can consider this graph to describe the path by which trust is propagated in the system.[2]

1. We say that a *direct* path exists from b to a if and only if b can prove (in the context of the agent) something about a that no other host can. That is, b has some *extra* information about a that others cannot extract from b's proof.

2. Let $\{h_0, h_1, \ldots h_n\}$ be a set of hosts for some $n \geq 1$. We say a *chained* path exists from h_n to h_0 if and only if there exists a direct path from h_x to h_{x-1} for each x from 1 to n.

[2] We intuitively define trust to propagate in the reverse direction of the agent. If the agent moves from a to b, we are interested to know if b trusts a. That is, if there is path from b to a. Moreover we are only interested in those hosts that modified the dynamic part.

3. We say that there is a *one-way* chained path from b to a if and only if there is a chained path from b to a and there is no (direct or chained) path from a to any other host.

Assume that i is the initiator of the agent, a is any sending host and b is the receiving host. Also, excepting the act of agent transfer no other interaction is allowed between any hosts. Using this scenario, authentication can be redefined in the context of b as follows:

(a) *Relative*: Determine that a chained path from a to i exists.

(b) *Absolute*: Determine that a one-way chained path from a to i exists.

2.4 Definitions

(a) Fix the alphabet $\Sigma = \{0, 1\}$. The set of strings over Σ is denoted by Σ^* and the set of all strings over Σ of length $\leq l$ are denoted by Σ^l. For any $x, y \in \Sigma^*$ the symbol $x \| y$ denotes the concatenation of x and y. The empty string is denoted by the symbol ϵ.

(b) If a, b are two variables of the same type, then $a \leftarrow b$ denotes that a is set to the value of b. If $\mathbb{A}$ is a non-empty set, then $x \leftarrow \mathbb{A}$ denotes that x has been uniformly chosen in $\mathbb{A}$. The symbol $|\mathbb{A}|$ denotes the cardinality of $\mathbb{A}$. Throughout this paper we will use the symbol $\mathbb{Z}$ to denote the set of integers and the symbol $\mathbb{I}$ to denote the set of all identities $\{I_1, I_2, \ldots\}$.

(c) We denote any (ordered) sequence of n elements $\alpha_1, \alpha_2, \ldots, \alpha_n$ symbolically by $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$. The *empty* sequence is a sequence without any elements and is denoted by $\langle\rangle$. If $S = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ is some finite sequence then $\langle S, \alpha_i \rangle = \langle \alpha_1, \alpha_2, \ldots, \alpha_n, \alpha_i \rangle$ is also a finite sequence. For any finite set $\mathbb{A}$, the symbol $\langle \mathbb{A} \rangle$ denotes the set of all sequences having (non-repeating) elements from $\mathbb{A}$.

(d) Fixed Strings: Let $\mathbb{L}_1$ and $\mathbb{L}_2$ be any two languages. For some $x \in \mathbb{L}_1$ and some $y \in \mathbb{L}_2$, the pair $\langle x, y \rangle$ is said to be *fixed* if and only if there exists a (polynomial-time computable) binary function $\sigma : \mathbb{L}_1 \times \mathbb{L}_2 \mapsto \{0, 1\}$ such that $\sigma(x, y) = 1$ and it is computationally intractable to find another string $\hat{y} \in \mathbb{L}_2$ such that $\sigma(x, \hat{y}) = 1$.

2.5 Bilinear Pairings

The fundamental building blocks of our protocol are a class of primitives known as *bilinear pairings*[3] defined as follows: Let $\mathbb{G}_1$ be a cyclic additive group of prime order q and $\mathbb{G}_2$ be a cyclic multiplicative group of the same order. Assume that computing the discrete logarithm in both $\mathbb{G}_1$ and $\mathbb{G}_2$ is hard. A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ that satisfies the following properties:

1. *Bilinearity*: $e(aP, bQ) = e(P, Q)^{ab} \quad \forall P, Q \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_q$
2. *Non-degeneracy*: $P \neq 0 \Rightarrow e(P, P) \neq 1$
3. *Computability*: e is efficiently computable

Typically, the map e will be derived from either the Weil or Tate pairing on an elliptic curve over a finite field. Despite the fairly complex mathematics involved in constructing such maps, cryptographic protocols based on pairings can be described entirely without ever referring to the actual implementation. Pairings and other parameters should be selected in proactive for efficiency and security. We refer the reader to [19, 11, 20] for details on generating secure parameters for such pairings. A more general definition of bilinear pairings (also suitable for our purpose) using three groups is a map $e : \mathbb{G}_0 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ such that an efficiently computable isomorphism $\mathbb{G}_1 \mapsto \mathbb{G}_0$ exists and the usual properties of bilinearity, non-degeneracy and computability are satisfied [19]. For simplicity in this paper we assume $\mathbb{G}_0 = \mathbb{G}_1$.

Our motivation to use pairings is due to the fact that the Decisional Diffie-Hellman problem (DDHP) in $\mathbb{G}_1$ is easy while both the Diffie-Hellman Problem (DHP) and the Discrete Logarithm Problem (DLP) are hard in $\mathbb{G}_1$ (see [19] for a proof). Such groups (where DDHP is easy but DHP is hard) are generally referred to as as *Gap* Diffie Hellman (GDH) groups [21].

[3] Bilinear pairings are probably best known for their use in Identity Based Encryption (IBE) by Boneh and Franklin in 2001 [11]. Many other applications of bilinear pairings are known. For example, various types of Identity Based Signatures (IBS) [13, 14, 15], tripartite one-round key agreement [16], Certificate-Less Public Key Cryptography (CL-PKC) [12], self-blindable credential certificates [17] and authenticated key agreement [18] are all based on pairings.

Our scheme is similar to the batch signature scheme presented in [19] or the aggregate signature schemes in [22] where signatures of many users (on different messages) are verified in one single step. In our scheme, signatures of many users on the same message are verified at once. Moreover in our scheme, the exact order of individual signers involved in creating the aggregate signatures is preserved.

3 Problem Formulation

In this section, we will define the problem of host protection using authentication primitives and set out the goals of our proposed authentication protocol. Although we consider only one agent in our analysis, this setup can also be used in a multi-agent system. We model our protocol on the following assumptions:

1. The mobile agent will be partitioned using an ideal partitioning scheme (see section 2.1). Without loss of generality we assume that the identities of the first n participants is the ordered sequence $\langle I_1, I_2, \ldots I_n \rangle$ where $I_i \in \mathbb{I}$ for each positive integer i. Represent by M, the static part and by D_i the dynamic part of the i^{th} instance. For any agent $\langle M, D_i \rangle$, the sending, platform is I_i and the receiving platform is I_{i+1} and the initiator is I_1. Agents are allowed to travel indefinitely and also to replicate and branch to different paths. The only restriction is that an agent must not return back to a past platform (except at the end of its itinerary).

2. To enable absolute authentication, we require that the pair $\langle M, I_1 \rangle$ be fixed (see section 2.4). A possible approach for this is to involve a Trusted Third Party (TTP) to certify this pair. The TTP ensures that the same pair cannot be reused again for a certain period of time. We note, however, that it may also be possible to implicitly fix the pair $\langle M, I_1 \rangle$ (without involving a TTP) using the methods for code obfuscation, undetachable signatures and watermarking described in [2, 4, 5, 23, 24, 25, 26, 9]. For simplicity, in this paper, we assume a TTP is used to fix the pair $\langle M, I_1 \rangle$.[4]

3. The mobile agent *must* always transferred with an accompanying signature. Each receiving platform I_i *must* verify the signature of the previous platform before it is executed. Execution should only be possible if the verification process succeeds and other security policies of the platform are satisfied.

4. Each sending platform I_i *must* sign the agent after it completes execution and before it is transferred. Moreover if this sending platform is not the first platform in the chain, it should sign the agent only if the verification process on the signature of the previous platform succeeded.

5. Each receiving platform would like to know the exact order of the platforms involved in passing (and executing) the agent. The purpose of the signature scheme is to ensure that the verification process succeeds if and only if the correct order of participants is given as input to the process. Any misbehavior (deviation from the signing or verification process) should be detected along with the concerned participant(s).

6. The itinerary of the agent is 'ad-hoc'. It is not possible for any platform I_i to determine the exact future itinerary of the agent (we can consider the agent to be autonomous in this case). Thus, a sending platform may not know the real identity of the next receiving platform. For simplicity, we assume that each sending platform I_i *does not* need to know the identity of the next receiving platform I_{i+1} at the time of signing.

7. Agent transfer is done over a secure channel where confidentiality is assured by the use of encryption. A Public Key Infrastructure (PKI) will be used for authentication (in the next section, we will describe this PKI). If needed, the same or a different PKI can be used for encryption.

4 Our Authentication Protocol

A one-time initial setup is necessary during which our participants create a public-key directory. Once this setup is complete, Any member can initiate an agent into the system. Members can also execute an agent and transfer agents to other members. Our protocol allows multi-hop agents to be authenticated.

[4] The concept of *liability* is worth mentioning here. In most cases, trust and liability go hand in hand: If Alice is trusted, she is liable if she fails the trust. An attacker will try to gain more trust but not liability. In the situation mentioned here, if the attacker removes all the names from the list and $\langle M, I_1 \rangle$ is not fixed, it may be possible that the attacker becomes automatically more liable (since the attacker's name cannot be removed from the list). We can safely ignore this possibility in applications where the liability of removing the names outweighs the the benefit gained from such an attack.

4.1 Initial Setup (Create PKI)

A public directory (or PKI) will be used to authenticate messages (and if necessary to encrypt them). The PKI we describe is based on bilinear pairings[5] and a trusted central authority is responsible for generating the security parameters. To participate in the protocol each user must have a certified public key (the process of certification is outside the scope of our protocol). The setup proceeds as follows:

1. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ be a bilinear mapping as defined in section 2.5. Let $P \in \mathbb{G}_1$ be a generator of $\mathbb{G}_1$ and let $H : \Sigma^* \mapsto \mathbb{G}_1$ and $h : \Sigma^* \mapsto \Sigma^l$ be two cryptographic hash functions. The parameters $\langle e, q, \mathbb{G}_1, P, H, h\rangle$ are generated by the trusted authority and made public in an authentic way.

2. Each participant I_i generates $x_i \leftarrow \mathbb{Z}_q$ as the private key. The corresponding public key is $Y_i = x_i P$

3. Each participant who wants to sign messages obtains a certificate from some trusted CA linking the identity I_i and the public key Y_i

This infrastructure can also be used to encrypt messages to any receiver I_j with the (certified or uncertified) public key x_jP using the protocol presented in [29] as follows: The sender will first encrypt the message with a symmetric cipher using the key derived from $K = r_j(x_jP)$ where $r_j \leftarrow \mathbb{Z}_q$. The sender will transmit the ciphertext along with the partial key $L = r_jP$ using an insecure public channel. Receiver I_j can compute the same key-derivation key $K = x_jL$ to decrypt the ciphertext. This protocol is secure if the Diffie-Hellman Problem (DHP) in $\mathbb{G}_1$ is hard [30].

4.2 Agent Initiation

As mentioned earlier, the initiator I_1 will use a TTP to fix the pair $\langle M, I_1\rangle$ to ensure that a different user cannot act as the initiator for the same message in the future. I_1 sends a request the the TTP to certify the pair $\langle M, I_1\rangle$. Denote by C the certificate from the TTP.

4.3 Transfer Protocol

An arbitrary participant I_i will process the agent as follows: On receiving it from I_{i-1}, it first follows the verification procedure. Before passing the agent (after execution) to I_{i+1}, it follows the signing procedure. The first participant I_1, however, only follows the signing procedure. As mentioned earlier, messages and signatures are send over a secure encrypted channel. Thus, an eavesdropper does not have access to the signature of I_i sent to I_{i+1}. In the definitions below we assume that `MESSAGE` denotes the agent which consists of both static and dynamic parts.

Notation

1. A correctly formed signature consists of the following components: a certificate `CERTIFICATE`, a list of identifiers `IDENTIFIER-LIST`, a signature on the static part `STATIC-PART`, a signature on the dynamic part `DYNAMIC-PART`.

2. The signing procedure `CREATE-CHAIN-SIGNATURE` takes three inputs: a valid message `MESSAGE`, a valid signature `OLD-SIGNATURE` and an identifier `IDENTIFIER`. It either outputs a new valid signature `NEW-SIGNATURE` or the error `ERROR`.

3. The verification procedure, `VERIFY-CHAIN-SIGNATURE` takes two inputs: a message `MESSAGE` and a signature `SIGNATURE` and outputs `TRUE` or `FALSE`.

4. Let $j \geq 0$, $k \geq 1$ be integers. Define the variables $t_j \in \Sigma^*, L_j \in \langle \mathbb{I}\rangle$ and $U_j, V_j \in \mathbb{G}_1$ as follows:

   (a) $t_0 = \epsilon$ the empty string, $L_0 = \langle\rangle$, the empty sequence and $U_0 = V_0 = 0$

   (b) $t_k = h(t_{k-1} \| h(I_k))$

   (c) $U_k = x_kH(M) + x_kH(t_k) + U_{k-1} = \sum_{r=1}^{r=k} x_r(H(M) + H(t_r))$

   (d) $V_k = x_jH(M\|D_k)$

   (e) $L_k = \langle L_{k-1}, I_k\rangle = \langle I_1, I_2, \dots, I_k\rangle$

[5] Although bilinear pairings are mostly known for their use in identity based cryptography, other non-identity based applications also have been proposed [19, 27, 28].

(A) `CREATE-CHAIN-SIGNATURE`

This procedure takes as input the `MESSAGE` $\langle M, D_i \rangle$, the `IDENTIFIER` I_i, the signature `OLD-SIGNATURE` and outputs `NEW-SIGNATURE` or `ERROR` where:

$$\texttt{OLD-SIGNATURE} = \langle C, L_{i-1}, U_{i-1}, V_{i-1} \rangle \text{ and } \texttt{NEW-SIGNATURE} = \langle C, L_i, U_i, V_i \rangle$$

We describe this procedure algorithmically:

1. Output `ERROR` if `OLD-SIGNATURE`, `MESSAGE` or `IDENTIFIER` has an has an invalid structure otherwise extract the following:
   (a) I_i from `IDENTIFIER`
   (b) $\langle M, D_i \rangle$ from `MESSAGE`
   (c) C from `CERTIFICATE` of `OLD-SIGNATURE`
   (d) $L_{i-1} = \langle I_1, I_2, \ldots, I_{i-1} \rangle$ from `IDENTIFIER-LIST` of `OLD-SIGNATURE`
   (e) U_{i-1} from `STATIC-PART` of `OLD-SIGNATURE`
2. Output `ERROR` if the private key x_i corresponding to I_i is not known otherwise compute the following:
   (a) $L_i = \langle L_{i-1}, I_i \rangle = \langle I_1, I_2, \ldots, I_i \rangle$
   (b) t_i from L_i using the algorithm: $t_0 \leftarrow \epsilon$, $t_j \leftarrow h(t_{j-1} \| h(I_j))$
   (c) $U_i = x_i H(M) + x_i H(t_i) + U_{i-1}$
   (d) $V_i = x_i H(M \| D_i)$
3. Set the following variables and output `NEW-SIGNATURE`:
   (a) `IDENTIFIER-LIST` $\leftarrow L_i$
   (b) `STATIC-PART` $\leftarrow U_i$
   (c) `DYNAMIC-PART` $\leftarrow V_i$
   (d) `NEW-SIGNATURE` $\leftarrow \langle$`CERTIFICATE`, `IDENTIFIER-LIST`, `STATIC-PART`, `DYNAMIC-PART`$\rangle$

(B) `VERIFY-CHAIN-SIGNATURE`

For clarity, we describe the verification procedure to be followed by I_{i+1}. This procedure takes as input the `MESSAGE` $\langle M, D_i \rangle$, the signature `SIGNATURE` and outputs `TRUE` or `FALSE`. The process can be described algorithmically:

1. Output `FALSE` if `SIGNATURE` or `MESSAGE` has an invalid structure otherwise extract the following:
   (a) $\langle M, D_i \rangle$ from `MESSAGE`
   (b) C from `CERTIFICATE` of `SIGNATURE`
   (c) $L_i = \langle I_1, I_2, \ldots, I_i \rangle$ from `IDENTIFIER-LIST` of `SIGNATURE`
   (d) U_i from `STATIC-PART` of `SIGNATURE`
   (e) V_i from `DYNAMIC-PART` of `SIGNATURE`
2. Output `FALSE` if any of the following checks fail otherwise output `TRUE`:
   (a) Check that C is valid for the pair $\langle M, I_1 \rangle$
   (b) Check that the sequence L_i does not contain any duplicate elements.
   (c) (i) Compute $\langle t_1, t_2 \ldots t_i \rangle$ using the algorithm $t_0 \leftarrow \epsilon$, $t_j \leftarrow h(t_{j-1} \| h(I_j))$
   (ii) Check that $e(U_i, P) \overset{?}{=} \prod_{r=1}^{r=i} e(H(M) + H(t_r), Y_r)$
   (d) Check $e(V_i, P) \overset{?}{=} e(H(M \| D_i), Y_i)$
   (e) Check that M and D_i belong to the same agent (via the mutually authenticating property)

If the output of the `VERIFY-CHAIN-SIGNATURE` process is `TRUE`, it can be ascertained (up to the level of trust placed on the TTP) that the itinerary proclaimed by the agent is correct. Execution privileges should be granted to the agent only if I_{i+1} trusts the TTP and *all* the identities in the itinerary to a satisfactory level. I_{i+1} can still choose to transfer the agent further even after denying execution privileges. Notice that our signatures are very similar to the aggregate signatures of Boneh et al. [22].

4.4 Correctness and Soundness

To prove the correctness and soundness of our scheme, we must first formally define our security goals and our adversary model. We want to ensure that the ordered list of participants specified in `IDENTIFIER-LIST` should correctly and uniquely identify the path of the received message.

(A) Correctness

We must show that if all the participants behave correctly, then the verification process will always succeed. The correctness of the verification process follows directly from the property of bilinear maps:

1. LHS of step 2(c) of verification process $= e(U_i, P)$

   $= e(\sum_{r=1}^{r=i} x_r(H(M) + H(t_r)), P) = \prod_{r=1}^{r=i} e(x_r(H(M) + H(t_r)), P)$

   $= \prod_{r=1}^{r=i} e(H(M) + H(t_r), x_r P) = \prod_{r=1}^{r=i} e(H(M) + H(t_r), Y_r)$

   $=$ RHS of step 2(c)

2. LHS of step 2(d) of verification process $= e(V_i, P)$

   $= e(x_i H(M||D_i), P) = e(H(M||D_i), x_i P)$

   $=$ RHS of step 2(d)

(B) Soundness

To prove soundness, we need to show that if any of the users misbehave, the verification will fail with a high probability. In other words, we must show that the scheme is existentially unforgeable with respect to the message M and each sequence L_i uniquely and correctly identifies the order of signers.

We note that the signatures V_i on the dynamic parts D_i are secure against existential forgery assuming the hardness of the BDHP as shown in [19]. Note that the value D_i is authenticated only up to the previous hop. The mutually authenticating property ensures that static and dynamic parts from different agents cannot be mixed. Our security analysis is therefore considered only with respect to the static part M.

1. Existential Unforgeability

We must show that our scheme is existentially unforgeable. In other words, it is not possible for any active adversary to forge signatures on any new (chosen) messages. This follows from the following theorem:

Theorem 1 *Assume that H is a random oracle and the Bilinear Diffie Hellman Problem (BDHP) in $(\mathbb{G}_1, \mathbb{G}_2)$ is hard.Then our scheme is secure against existential forgery.*

Proof: We assume that the hash function H is a random oracle. Define a family of hash functions $H_Z : \Sigma^* \mapsto \mathbb{G}_1$ as: $H_Z(X) = H(X) + Z$ where $Z \leftarrow \mathbb{G}_1$. Each function H_Z also behaves like a random oracle. Let $Z = H(M)$ and $X = t_i$.

In this notation, each U_i can be considered as an aggregation of individual signatures of I_j on t_j $\forall j \leq i$ using the aggregate signature scheme of [22] and setting the hash function as $H_{H(M)}$ for some known constant M because: $U_i = \sum_{r=1}^{r=i} x_r(H(M) + H(t_r)) = \sum_{r=1}^{r=i} x_r H_{H(M)}(t_r)$. Also since H is a random oracle, it is hard to find another value $M_1 \neq M$ such that $H(M) = H(M_1)$.

Assuming that $I_i \neq I_j$ whenever $i \neq j$ and $h(I_i) \neq \epsilon$ $\forall i$, it is also ensured that $t_i \neq t_j$ whenever $i \neq j$ (in other words, all t_i are distinct). It is shown in theorem 3.2 of [22] that this aggregate signature scheme is secure against existential forgery if the messages t_i are all distinct and the BDHP in (G_1, G_2) is hard. This completes the proof of security against existential forgery.

2. Chained Signature Unforgeability

To prove that our chained signature scheme is secure, it remains to be shown that each $t_i \in \Sigma^l$ corresponds to the unique sequence $L_i \in \langle \mathbb{I} \rangle$ or alternatively the pairs $\langle t_i, L_i \rangle$ are fixed since otherwise an attacker might be able to delete or change order of some identities in the list. Our proof is based on the hardness of inverting the hash function h. Recall that the range of h is Σ^l, the set strings all strings over

Σ of length $\leq l$. Before we proceed, we need the following result:

Theorem 2 *For any random string $a \in \Sigma^*$, if there is an oracle $\mathcal{Q}$ that can compute two strings $u, v \in \Sigma^*$ such that $a = (u \| h(v))$ and the length of $h(v)$ is uniformly chosen between 1 and k for some $k \leq l$ then h can be inverted for at least 2^k instances with non-negligible probability.*

Proof: Let b be a random string of at most k bits. We will use the oracle to invert h by finding a value c such that $b = h(c)$. Construct the string $a = (\text{"1"} \| b)$. The string a is given repeatedly as input to the oracle which will return different values of $\langle u, v\rangle$ for each query such that length of $h(v)$ is uniformly chosen between 1 and k and is independent of previous queries. For any query, $Pr[h(v) \neq b] = \frac{k-1}{k}$ and is independent of other queries. Thus, after n queries, $Pr[h(v) \neq b \ \forall n] = (\frac{k-1}{k})^n$ and this value can be made extremely low by choosing large enough n. On the other hand, if $h(v) = b$ then we have inverted h and found a value $c = v$ such that $b = h(c)$, thus completing the proof.

Corollary 1: If h is non-invertible for all instances then such an oracle $\mathcal{Q}$ cannot exist.

Assume that there is an adversary $\mathcal{A}$ who is able to compute a sequence $L_{\mathcal{A}} \in \langle \mathbb{I} \rangle$ such that $L_{\mathcal{A}} \neq L_i$ but t_i corresponds to $L_{\mathcal{A}}$. Also let $L_{\mathcal{A}} = \left\langle I_{\alpha_1}, I_{\alpha_2}, \ldots, I_{\alpha_j} \right\rangle$ and $L_i = \langle I_1, I_2, \ldots, I_i \rangle$. A successful attack by $\mathcal{A}$ will imply that $h(t_{\alpha_{j-1}} \| h(I_{\alpha_j})) = h(t_{i-1} \| h(I_i))$. Since h is a random oracle and is collision resistant, we will trivially assume that $(t_{\alpha_{j-1}} \| h(I_{\alpha_j})) = (t_{i-1} \| h(I_i))$ which leads to the following two possibilities:

1. $I_{\alpha_j} \neq I_i$ and $t_{\alpha_{j-1}} \neq t_{i-1}$
2. $I_{\alpha_j} = I_i$ and $t_{\alpha_{j-1}} = t_{i-1}$

The first possibility is ruled out due to corollary 1 since otherwise, we can simulate a call to the oracle $\mathcal{Q}$ by challenging $\mathcal{A}$ with the value $a = (t_{i-1} \| h(I_i))$. We will therefore only consider the second possibility in our analysis.

From the second possibility, $t_{\alpha_{j-1}} = t_{i-1}$ or alternatively, $h(t_{\alpha_{j-2}} \| h(I_{\alpha_{j-1}})) = h(t_{i-2} \| h(I_{i-1}))$. Using the induction hypothesis and the above analogy, we see that one of the following must hold:

1. $i > j$. Therefore $h(h(I_{\alpha_1})) = h(t_{i-j} \| h(I_{i-j+1}))$ and $t_{i-j} \neq \epsilon$
   Assume that h is a random oracle and collision resistant. We then have, $h(I_{\alpha_1}) = (t_{i-j} \| h(I_{i-j+1}))$. Alternatively, $I_{\alpha_1} = h^{-1}(t_{i-j} \| h(I_{i-j+1})$. Computing such an I_{α_1} requires the ability to invert h so this case is ruled out.

2. $i < j$. Therefore $h(t_{\alpha_{j-i}} \| h(I_{\alpha_{j-1+1}})) = h(h(I_1))$ and $t_{\alpha_{j-i}} \neq \epsilon$.
   Assume that h is collision resistant and thus, $h(I_1) = (t_{\alpha_{j-i}} \| h(I_{\alpha_{j-1+1}}))$. Setting $a = h(I_1)$, we see from theorem 2 and corollary 1 that computing such an $I_{\alpha_{j-1+1}}$ is not feasible and hence this case is also ruled out.

3. $i = j$. Therefore $I_{\alpha_i} = I_i$ and $t_{\alpha_i} = t_i \ \forall i$
   Alternatively this implies that $L_{\mathcal{A}} = L_i$. This is a contradiction to our assumption that $L_{\mathcal{A}} \neq L_i$ and therefore such an adversary $\mathcal{A}$ cannot exist. This completes our proof of security.

5 Overview of the protocol

The above protocol is an example of a one-way signature chaining scheme. To understand this, see that step 7 of the verification process involves the public keys of all participating users (in the right order). Moreover, since M and I_1 cannot be unlinked due to the certificate C, it is ensured that a different initial user cannot be used for M.

We see that the signatures have an "additive" property, demonstrated by the fact that I_{i+1} can 'add' more information to the signature U_i of I_i by computing U_{i+1}. Note that computing any U_i just from U_{i+1} is considered infeasible due to the assumed properties of the bilinear map.[6] User I_{i+1} sends U_{i+1} as part of the new signature while it keeps U_i from the old (received) signature as its secret evidence in case of a dispute. Non-repudiation is provided as follows: (Note that I_{i+1} must have saved the entire signature of I_i). I_{i+1} can prove in a court that the message was indeed received from I_i by producing this

[6] Observe that U_i cannot be computed from U_{i+1} without knowledge of x_i but knowledge of U_i does not reveal x_i.

signature as a *witness* and invoking the `VERIFY-CHAIN-SIGNATURE` procedure. Assuming that all users are unique, a few points about this protocol are noteworthy:

1. Each I_i who passes the message must include its name in the signature and in the right sequence for validation to succeed.

2. Users cannot remove names of other users from the list in the signature without knowledge of their private keys, nor can they change the order or add new names.

3. The dynamic part is only authenticated to the previous hop. The itinerary authentication is done entirely using the static part. Also, authentication is relative to I_1 who in turn authenticates with the TTP. If, however, it is possible to establish the originator of a message directly from its contents or by some other means, the TTP can be eliminated. For a discussion on this see [9].

4. The signing and verification procedures are completely non-interactive. Moreover, it is possible to combine the signing and verifying procedures into a single *sign-verify* procedure to increase efficiency. However, there will always be a temporal ordering with verification and signing in our mobile agent scenario (corresponding to before and after the execution of the mobile agent).

5. The signing process requires two multiplications and one addition in $\mathbb{G}_1$. The verification process requires $\mathcal{O}(n)$ multiplications in $\mathbb{G}_2$ and $\mathcal{O}(n)$ pairing computations. It is easily seen that the signing time is independent of the number of users. Also, the signature size is constant ignoring the payload of the identifier list (which cannot be avoided).

Our protocol demonstrates a type of chaining called *backward* chaining where each receiver of the message is responsible for "adding" a link to the chain. Likewise, we can also consider *forward* chaining where the senders of the message are responsible for creating the chain. In this variant, each sender is aware of the next receiver during the signing process. Forward chaining has the advantage that the order of participants can be strictly specified by senders. However, such a scheme also restricts the flexibility of the system because the message will have to be signed multiple times if sent to many receivers in parallel. Moreover in a backward chaining scheme, multiple senders within a 'trust zone' can use a single signing gateway without revealing the identity of the recipients. Due to these reasons, we only considered backward chaining in our work.[7]

6 Other Applications of Signature Chaining

In this section, we list several applications of signature chaining. If we consider the message without the dynamic part, we get a simple signature-chaining protocol for message passing, with the message being M, the static part. For all the applications discussed in this session, we assume that `MESSAGE` is simply the static part and the signing and verification procedures and the signature structure are accordingly modified to exclude all references to the dynamic part (in other words, steps 2(d) and 3(c) of the signing process and steps 1(e), 2(d) and 2(e) of the verification process are excluded).

Considering that one-way signature chaining enables us to correctly validate the path of any received message and provides non-repudiation, we can consider several applications: electronic auctions, payment systems [31], group e-commerce (e-commerce transactions where multiple entities are involved such that direct interaction is not possible between many of them), electronic work-flow enforcement (ensuring the order in which participants should be involved),'secret-passing' protocols, secure routing, authenticated mail relaying and spam tracing, token based authentication, IP tracing, mobile IP, intrusion detection, GRID computing, battlefield modeling, Supply Chain Management, distributed systems and wireless roaming.

7 Conclusion and Future Directions

In this paper, we proposed an authentication protocol for mobile agents based on bilinear pairings over elliptic curves. Our method is based on the notion of additive zero knowledge [9] which enables trust

[7] Forward chaining is easy to construct but inefficient in practice. Each signer I_i simply signs the value $(M\|I_{i+1})$ using any ordinary digital signature scheme. The chained-signatures of I_i is the set of signature of all the previous signers including this signature.

to propagate between different provers. We demonstrated that signature chaining can be used to form ad-hoc trust relationships between multiple participants in a dynamic and non-interactive manner. Our protocol can be used to authenticate the itinerary of mobile agents without any active involvement of a Trusted Third Party(TTP). It may be possible to completely eliminate the TTP using methods of code obfuscation, watermarking and undetachable signatures.

Our protocol uses a standard certificate-based PKI and it is worth researching if a certificate-less or an identity based scheme can be derived from the certificate based one presented in this paper. The other aspect of the paper described the concept of agent partitioning (section 2.1). It is an open question if a secure and ideal partitioning scheme can be constructed for mobile agents. However, it seems plausible considering the recent developments in java bytecode verifiers [32, 33, 34, 35, 36].

References

[1] David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *SIGOPS Oper. Syst. Rev.*, 33(3):7–13, 1999.

[2] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science*, 1419:44–60, 1998.

[3] Joy Algesheimer, Christian Cachin, Jan Camenisch, and Günter Karjoth. Cryptographic security for mobile code. In *SP '01: Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–11. IEEE Computer Society, 2001.

[4] Panayiotis Kotzanikolaou, Mike Burmester, and Vassilios Chrissikopoulos. Secure transactions with mobile agents in hostile environments. In *Australasian Conference on Information Security and Privacy*, volume 1841, pages 289–297, Australia, 2000. Springer-Verlag.

[5] Joris Claessens, Bart Preneel, and Joos Vandewalle. (how) can mobile agents do secure electronic transactions on untrusted hosts? a survey of the security issues and the current solutions. *ACM Trans. Inter. Tech.*, 3(1):28–48, 2003.

[6] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.

[7] U. G. Wilhelm, S. Staamann, and L. Buttyán. Introducing trusted third parties to the mobile agent paradigm. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603, pages 471–491. Springer-Verlag, New York, NY, USA, 1999.

[8] G. Karjoth, D.B. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, 1(4):68–77, 1997.

[9] Amitabh Saxena and Ben Soh. Authenticating mobile agent platforms using signature chaining without trusted third parties. In *Proceedings of The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05)*, pages 282–285, Hong kong, 2005. IEEE computer press.

[10] Amitabh Saxena and Ben Soh. A novel method for authenticating mobile agents with one-way signature chaining. In *Proceedings of The 7th International Symposium on Autonomous Decentralized Systems (ISADS 05)*, pages 187–193, China, 2005. IEEE Computer Press.

[11] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229. Springer-Verlag, 2001.

[12] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. Cryptology ePrint Archive, Report 2003/126, 2003.

[13] Kenneth G. Paterson. Id-based signatures from pairings on elliptic curves. Cryptology ePrint Archive, Report 2002/004, 2002.

[14] Song Han, Winson K.Y. Yeung, and Jie Wang. Identity-based confirmer signatures from pairings over elliptic curves. In *EC '03: Proceedings of the 4th ACM conference on Electronic commerce*, pages 262–263, New York, NY, USA, 2003. ACM Press.

[15] Amit K Awasthi and Sunder Lal. Id-based ring signature and proxy ring signature schemes from bilinear pairings. Cryptology ePrint Archive, Report 2004/184, 2004.

[16] Antoine Joux. A one round protocol for tripartite diffie-hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[17] Eric R. Verheul. Self-blindable credential certificates from the weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–551, London, UK, 2001. Springer-Verlag.

[18] N. Smart. An identity based authenticated key agreement protocol based on the weil pairing. Cryptology ePrint Archive, Report 2001/111, 2001.

[19] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001. Springer-Verlag.

[20] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, London, UK, 2002. Springer-Verlag.

[21] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 104–118, London, UK, 2001. Springer-Verlag.

[22] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[23] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28(8):735–746, August 2002.

[24] Chenxi Wang, Jonathan Hill, John Knight, and Jack Davidson. Software tamper resistance: Obstructing static analysis of programs. Technical report, University of Virginia, University of Virginia, 2000.

[25] Julien P. Stern, Gael Hachez, Francois Koeune, and Jean-Jacques Quisquater. Robust object watermarking: Application to code. In *Information Hiding*, volume 1768, pages 368–378, Germany, 1999. Springer-Verlag.

[26] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. Cryptology ePrint Archive, Report 2001/069, 2001.

[27] Z. Cheng, L. Vasiu, and R. Comley. Pairing-based one-round tripartite key agreement protocols, 2004.

[28] Ratna Dutta, Rana Barua, and Palash Sarkar. Pairing-based cryptographic protocols : A survey. Cryptology ePrint Archive, Report 2004/064, 2004.

[29] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[30] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, Norwell, MA, USA, 1994. Foreword By-Neal Koblitz.

[31] Amitabh Saxena, Ben Soh, and Dimitri Zantidis. A digital cash protocol based on additive zero knowledge. In *Proceedings of The 3rd International Workshop on Internet Communications Security (ICCSA 05)*, volume 3482 of *Lecture Notes in Computer Science*, pages 672–680, Singapore, 2005. Springer-Verlag.

[32] Xavier Leroy. Java bytecode verification: Algorithms and formalizations. *J. Autom. Reason.*, 30(3-4):235–269, 2003.

[33] Pieter H. Hartel and Luc Moreau. Formalizing the safety of java, the java virtual machine, and java card. *ACM Comput. Surv.*, 33(4):517–558, 2001.

[34] C. League, V. Trifonov, and Z. Shao. Functional java bytecode. In *In: Proc. 5th World Conf. on Systemics, Cybernetics, and Informatics. (2001) Workshop on Intermediate Representation Engineering for the Java Virtual Machine.*, 2001.

[35] A. Coglio. Simple verification technique for complex java bytecode subroutines. In *In: Proc. 4th ECOOP Workshop on Formal Techniques for Javalike Programs. 39*, 2002.

[36] A. Coglio. Improving the official specification of java bytecode verification. In *Proceedings of the 3rd ECOOP Workshop on Formal Techniques for Java Programs, June 2001.*, 2001.