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

August 28, 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.

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 only 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 Fixed Strings

Let L_1 and L_2 be any two languages. For some $x \in L_1$ and some $y \in L_2$, the ordered pair (x, y) is said to be *fixed* if and only if there exists a (polynomial-time computable) binary function $\sigma : L_1 \times L_2 \mapsto \{0, 1\}$ such that $\sigma(x, y) = 1$ and it is computationally intractable to find another string $\hat{y} \in L_2$ 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] Let $\mathbb{G}_1$ be a cyclic additive group generated by P, whose order is a prime 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ and satisfies the following properties:

1. *Bilinearity*: $e(aP, bQ) = e(P, Q)^{ab}$ For all $P, Q \in \mathbb{G}_1$ 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 $e : \mathbb{G}_1 \times \mathbb{H}_1 \mapsto \mathbb{G}_2$ such that an efficiently computable isomorphism $\mathbb{H}_1 \mapsto \mathbb{G}_1$ exists and the usual properties of bilinearity, non-degeneracy and computability are satisfied [19]. For simplicity in this paper we assume $\mathbb{G}_1 = \mathbb{H}_1$.

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 [11] for a proof). This enables us to create proxy signatures in $\mathbb{G}_1$ that can be validated due to the tractability of the DDHP. Such groups (where DDHP is easy but DHP is hard) are generally referred to as as *Gap* Diffle Hellman (GDH) groups [21]. Our scheme is similar to the batch signature scheme presented in [19] where signatures of many users (on the same or different message) are verified in one single step. In our scheme, signatures of many users on the same message are verified at once. The 'batch' verification process, however, takes as input only one constant size 'aggregte' signature instead of a batch of signatures. 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:

[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.

1. The mobile agent can be partitioned using an ideal scheme (see section 2.1). Represent by M, the static part and by D_i the dynamic part of the i_{th} instance. For any agent $\{M, D_i\}$, the sending, platform is I_i and the receiving platform is I_{i+1}. The initiator of the agent is I_0

2. To enable absolute authentication, we require that the pair (M, I_0) 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 (M, I_0) (without involving a TTP) using the methods for code obfuscation, undetachable signatures and watermarking described in [2, 4, 5, 22, 23, 24, 25, 9]. For simplicity, in this paper, we assume a TTP is used to fix the pair (M, I_0).[4]

3. Agents can replicate and the same instance may be sent to many receivers. There is no limit to the number of times an agent may be transferred. The only restriction is that an agent must not return back to a past platform. The exception to this is when the agent returns back to the originator at the end of its itinerary.

4. 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.

5. 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.

6. 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).

7. 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.

8. 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. First we give some more notation: If A is a non-empty set, then $x \leftarrow A$ denotes that x has been uniformly chosen in A. If x and y are two strings then the symbol $x\|y$ denotes the concatenation of x and y.

4.1 Initial Setup (Create PKI)

In this section we describe how to setup a public directory (or PKI) that will be used to authenticate messages (and if necessary to encrypt them). The PKI we describe is based on bilinear pairings.[5] A trusted central authority is responsible for creating the PKI. To participate in the authentication protocol,

[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 (M, I_0) 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.

[5] Although bilinear pairings are mostly known for their use in identity based cryptography, other non-identity based applications also have been proposed [19, 26, 27]. Our authentication protocol presented here is based on an ordinary certificate based PKI.

each user must have a certified public key (We consider the process of certification outside the scope of our protocol). The setup protocol 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 be a generator of $\mathbb{G}_1$. Let $H : \{0,1\}^* \mapsto \mathbb{G}_1$ be a cryptographic hash functions. The parameters $\langle e, q, \mathbb{G}_1, P, H \rangle$ 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 using the (certified or uncertified) public key $x_j P$ as follows: The sender will first encrypt the message with a symmetric cipher using the key derived from $K_j = r_j(x_j P)$ where $r_j \leftarrow \mathbb{Z}_q$. The sender will transmit the ciphertext along with the partial key $L_j = r_j P$ using an insecure public channel. Receiver I_j can compute the same key-derivation key $K_j = x_j L_j$ to decrypt the ciphertext. This protocol is secure if the Diffie-Hellman Problem (DHP) in $\mathbb{G}_1$ is hard.

4.2 Agent Initiation

As mentioned earlier, the initiator I_0 will use a TTP to fix the pair $\{M, I_0\}$ to ensure that a different user cannot act as the initiator for the same agent later on. It asks the TTP to certify the ordered pair (M, I_0) using a standard signature scheme (like RSA). Denote by C, the certificate from the TTP. To avoid chosen ciphertext attacks, a time stamp is included in the certificate. Users who created their public keys after this time are precluded from participating in this protocol.

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 and aborts if it fails. Before passing the agent (after execution) to I_{i+1}, it follows the signing procedure.[6] The first participant I_0, however, only follows the signing procedure. As mentioned earlier, we assume that 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}.

Notation

In the definitions below we assume that `MESSAGE` denotes the agent which consists of both static and dynamic parts.

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`. We assume that the current user's private key is implicitly given as input to the signing function. Since I_0 is the first participant, it invokes the signing function with an empty `IDENTIFIER-LIST`.
3. The verification procedure, `VERIFY-CHAIN-SIGNATURE` takes two inputs: a message `MESSAGE` and a signature `SIGNATURE` and outputs `TRUE` or `FALSE`.
4. (a) Define $U_0 = x_0 H(M)$
   (b) Define $U_i = (i+1)x_i H(M) + U_{i-1}$ if $i > 0$
   (c) Define $Z_i = x_i H(D_i \| M)$

   Thus $U_i = \sum_{r=0}^{r=i} (r+1) x_r H(M)$

[6] We observe that it is possible to combine the signing and verifying procedures into a single *sign-verify* procedure. However, there will always be a temporal ordering with verification and signing in our case (corresponding to before and after the execution of the mobile agent)

(A) `CREATE-CHAIN-SIGNATURE`

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

`OLD-SIGNATURE` $=\{C, \{I_0, I_1, \ldots, I_{i-1}\}, U_{i-1}, Z_{i-1}\}$ and

`NEW-SIGNATURE` $=\{C, \{I_0, I_1, \ldots, I_i\}, U_i, Z_i\}$

We describe this procedure algorithmically:

1. Output `ERROR` if `OLD-SIGNATURE`, `MESSAGE` or `IDENTIFIER` has an has an invalid structure.
2. Extract I_i from `IDENTIFIER` and extract $\{M, D_i\}$ from `MESSAGE`. Output `ERROR` if the private key x_i corresponding to I_i is not known.
3. Extract C from `CERTIFICATE` of `OLD-SIGNATURE`
4. If $(i > 0)$ Extract $\{I_0, I_1, \ldots, I_{i-1}\}$ from `IDENTIFIER-LIST` of `OLD-SIGNATURE`
5. If $(i > 0)$ Extract U_{i-1} from `STATIC-PART` of `OLD-SIGNATURE`
6. Create `IDENTIFIER-LIST` $= \{I_0, I_1, I_2, \ldots, I_i\}$
7. Compute $U_i = (i+1)x_i H(M)$
8. If $(i > 0)$ Set $U_i \leftarrow U_i + U_{i-1}$
9. Create `STATIC-PART` $= U_i$
10. Create `DYNAMIC-PART`, $Z_i = x_i H(D_i \| M)$
11. Output `NEW-SIGNATURE` $=\{$`CERTIFICATE`, `IDENTIFIER-LIST`, `STATIC-PART`, `DYNAMIC-PART`$\}$

(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` $\{M, D_i\}$, 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.
2. Extract $\{M, D_i\}$ from `MESSAGE`
3. Extract C from `CERTIFICATE` of `SIGNATURE` and obtain I_0. Verify C for M and I_0. Output `FALSE` if verification fails
4. Extract $\{I_0, I_1, I_2, \ldots, I_i\}$ from `IDENTIFIER-LIST` of `SIGNATURE`
5. Extract U_i from `STATIC-PART` of `SIGNATURE`
6. Extract Z_i from `DYNAMIC-PART` of `SIGNATURE`
7. Verify: $e(U_i, P) \stackrel{?}{=} e(\sum_{r=0}^{r=i}(r+1)Y_r, H(M))$. Output `FALSE` if the check fails
8. Verify: $e(Z_i, P) \stackrel{?}{=} e(Y_i, H(D_i \| M))$. Output `FALSE` if the check fails
9. Verify that M and D_i belong to the same agent (via the mutually authenticating property). Output `FALSE` if the check fails.
10. Output `TRUE`

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 the signatures U_i are very similar to the short signatures proposed by Boneh, Shacham and Lynn in [19].

4.4 Correctness and Soundness

In this section, we outline a rough security analysis of our protocol. We consider an attack to be successful if the ordered list of names in the signature contains false information and the verification procedure accepts. Assuming that I_i is the attacker, a combination of the following attacks are possible:

1. It does not include its name in the list.
2. It adds one or more names to the list.
3. It deletes one or more names from the list or changes the order of names.

We will consider each scenario separately. We note that a detailed security analysis of the protocol is out of the scope of this paper but we also note that the simplicity of the protocol does not demand such analysis.

1. The first possibility is ruled out since otherwise step 7 and 8 of the verification process will simultaneously fail.
2. Arbitrary names cannot be added to the list because I_i cannot compute signatures M_i on behalf of other users. Thus, if a false user is added to the list, step 7 of the verification process will fail.
3. Finally deleting names or changing order is not possible either. If the order of participants is changed, the verification process in step 8 will fail with a very high probability.

We enumerate the following characteristics of our scheme:

1. Signature Unforgeability: It is not possible for any participant to generate signatures $U_{(.)}$for other participants without knowledge of their private keys assuming the hardness of the Bilinear Diffie-Hellman problem (BDHP). Similarly computing any private keys from the public information is will be equivalent to solving the Discrete Logarithm (DL) problem in $\mathbb{G}_1$ (and consequently $\mathbb{G}_2$).
2. Chained Signature Unforgeability: Similarly it is hard to add arbitrary participants in the chained signatures without knowledge of their public key due to the difficulty of the DL problem.

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_0 cannot be un-linked 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.[7] 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 `SIGNATURE` of I_i). I_{i+1} can prove in a court that the message `MESSAGE` was indeed received from I_i by producing this signature as a *witness* and running 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. Authentication is relative to I_0 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].

[7] 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.

4. The signing and verification procedures are completely non-interactive.

5. The dynamic part is only authenticated to the previous hop. The itinerary authentication is done entirely using the static part.

6. The signing process requires two elliptic curve point multiplications. The verification process requires $\mathcal{O}(n^2)$ point additions and two pairing computations. It is easily seen that the signing time is independent of the number of users and verification time is almost constant (assuming that point addition is very efficient in comparison to point multiplication). Moreover the signature size is constant ignoring the payload of the identifier list (which cannot be avoided).

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 the next 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, step 10 of the signing process and steps 5, 8 and 9 of the verification process are excluded).

The above protocol demonstrates a type of chaining called *backward* chaining where each receiver of the message is responsible for "adding" a link to the chain. 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 denders 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.[8]

6 Applications of Signature Chaining

In this section, we list several applications of signature chaining. The concept of signature chaining was originally proposed for mobile agent authentication [9, 10], electronic auctions, proxy signatures [28] and digital cash [29] but without any practical examples.

Considering that one-way signature chaining enables us to correctly validate path of any received message and provides non-repudiation, we can consider various other applications: 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 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 [30, 31, 32, 33, 34, 35, 36].

[8] 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.

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] 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.

[23] 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.

[24] 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.

[25] 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.

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

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

[28] Chunbo Ma. Proxy chain signature. Unpublished Manuscript, 2005.

[29] 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.

[30] X. Leroy. Java bytecode verification: algorithms and formalizations. *Journal of Automated Reasoning, 2003. To appear.*, 2003. To appear.

[31] 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.

[32] 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.

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

[34] 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.

[35] Gerwin Klein. *Verified Java Bytecode Verification*. PhD thesis, Institut für Informatik, Technische Universität München, 2003.

[36] X. Leroy. Bytecode verification for java smart card. *Software Practice & Experience*, 32.