# Additive Zero-Knowledge and Applications: SPAM Prevention

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

Chunbo Ma
School of Information Science and Technology
Southwest Jiaotong University
Chengdu, Sichuan, 610031, P. R. China

*Abstract*— **In this paper, we introduce the concept of *additive* zero knowledge. Essentially, an additive proof can be considered as a proof system involving many provers and one verifier such that the statements of all the provers are proved simultaneously. Our model of additive proofs is presented using constructions of *blind group identification*, *aggregate signatures* and *chained signatures*. The security of our protocols relies on the difficulty of the underlying Diffie-Hellman problem in bilinear maps. As applications, we present a novel method to prevent Spam.**

## I. Introduction

An additive proof is a non-interactive proof involving many provers and one verifier such that the verifier process accepts if and only if *all* the provers are honest. The 'zero-knowledge' property requires that other than this fact, no 'extra' information is revealed to the verifier. We use the following notation:

1) A **statement** is any claim that can be True or False. For example $s =$ "I know $x$ corresponding to $y$" is a statement. We denote the set of all statements by $\mathbb{S}$.
2) Let $s \in \mathbb{S}$ and $p \in \{0,1\}^*$. We say that the tuple $\langle s, p\rangle$ constitutes a **non-interactive proof system** (NIPS or simply PS from now onwards) if there is an efficiently computable mapping $\mathit{Ver}_1 : \mathbb{S} \times \{0,1\}^* \mapsto \{0,1\}$ such that $\mathit{Ver}_1(s,p) = 1$ if and only if $s$ is True. Analogously, we say that a statement $s$ is True if and only if $\exists p \in \{0,1\}^* : \mathit{Ver}_1(s,p) = 1$.
3) A **composite statement** is simply a collection of statements combined using the logical conjunction ($\wedge$) operation. A composite statement is True if and only if all its constituent statements are simultaneously True.
4) Let $S \subset \mathbb{S}$ be some composite statement let and $p \in \{0,1\}^*$ be some string. We say that the tuple $\langle S, p\rangle$ constitutes a (non-interactive) **composite proof system** (CPS) if and only if there is an efficiently computable mapping $\mathit{Ver}_2 : 2^{\mathbb{S}} \times \{0,1\}^* \mapsto \{0,1\}$ such that $\mathit{Ver}_2(S,p) = 1$ if and only if $\forall s_i \in S$, $s_i$ is True. Note that $\mathit{Ver}_2(\{s\},p) = \mathit{Ver}_1(s,p)$.
5) $\langle S, p\rangle$ is a **composite zero-knowledge proof system** (CZKPS) if and only if $\langle S, p\rangle$ is a CPS and the verifier cannot later use the proof $p$ to convince anyone (including himself) about the validity of the individual constituent statements of $S$ independently of the others. Formally, given $\langle S, p\rangle$ it is infeasible to compute a pair $\langle S_i, p_i\rangle \neq \langle S, p\rangle$ such that $\mathit{Ver}_2(S_i, p_i) = 1 \wedge S_i \subseteq S$.
6) $\langle S, p\rangle$ is an **additive zero-knowledge proof system** (AZKPS) if and only if $\langle S, p\rangle$ is a CZKPS and given $\langle S, p\rangle$, an efficient algorithm exists to compute a tuple $\langle S_i, p_i\rangle$ such that:
$\mathit{Ver}_2(S_i, p_i) = 1 \wedge S_i = S \cup \{s_i\} \wedge s_i \notin S \wedge s_i$ is True $\wedge \langle S_i, p_i\rangle$ is also a CZKPS.
It may be possible that each such resulting CZKPS $\langle S_i, p_i\rangle$ is also an AZKPS. These are the type of additive proofs we will be interested in.
7) Finally, we differentiate between two types of composite proof systems: (a) **parallel** in which the individual statements can be supplied to $\mathit{Ver}_2$ in any order and (b) **sequential** in which the individual statements must be supplied in a specific order.

To give a meaningful context to additive proofs, we will present three distinct examples:

1) *Blind Group Identification*: A group of users can identify themselves to a server such that: (a) if all users are honest the server always accepts and (b) If any users are dishonest the server always rejects. However, in this case it is impossible to find out the actual identity of the particular cheating user(s). We thus coin the term 'blind' identification to this process.
2) *Aggregate Signatures*: A group of users will sign individual messages and combine the individual signatures to get an aggregate signature such that the verification process on the aggregate signature succeeds if and only if (a) all the individual signatures were valid and (b) the exact list of users (ignoring the order) involved in the aggregation is input to the verification process. Moreover, once the aggregate signature is created, it is impossible to extract any individual signatures.
3) *Chained Signatures*: These are exactly like aggregate signatures with the additional requirement that the exact order of the individual signers must be supplied to the verification process to make it accept a valid signature.

Throughout this paper, we will use the following notation. If $\mathbb{A}$ is a non-empty set, then $x \leftarrow \mathbb{A}$ denotes that $x$ has been uniformly chosen in $\mathbb{A}$. An *ordered* sequence of $i$ elements $\alpha_1, \alpha_2, \ldots \alpha_i$ is denoted by $\langle \alpha_1, \alpha_2, \ldots \alpha_i\rangle$. If $S = \langle \alpha_1, \alpha_2, \ldots \alpha_i\rangle$ is some finite sequence then $\langle S, \alpha\rangle = \langle \alpha_1, \alpha_2, \ldots \alpha_i, \alpha\rangle$ is also a finite sequence. For any two sequences $S = \langle \alpha_1, \alpha_2, \ldots \alpha_i\rangle$ and $T = \langle \beta_1, \beta_2, \ldots \beta_j\rangle$,

$S = T \Leftrightarrow (i = j \ \&\ \alpha_i = \beta_i \ \forall i)$.

The rest of the paper is organized as follows. In section II we present the cryptographic primitives used in our protocol. In section III, we describe the underlying Public Key Infrastructure required for our protocols. We present the group identification protocol in sections IV. Finally, in section V and VI, we present the aggregate signature and chained signature protocols along with a brief security analysis.

## II. Bilinear Pairings

All the constructions presented in this paper are based on the existence of efficiently computable non-degenerate bilinear maps which can be abstractly described 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]:

1) *Bilinearity*: $e(aP, bQ) = e(P, Q)^{ab} \ \forall P, Q \in \mathbb{G}_1$ 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

The above properties also imply:

$$e(P + Q, R) = e(P, R) \cdot e(Q, R) \ \forall P, Q, R \in \mathbb{G}_1$$
$$e(P, Q + R) = e(P, Q) \cdot e(P, R) \ \forall P, Q, R \in \mathbb{G}_1$$

Typically, the map $e$ will be derived from either the Weil or Tate pairing on an elliptic curve over a finite field. Despite the complex mathematics involved in constructing such maps, cryptographic protocols based on pairings can be described entirely without ever referring to the actual implementation. We refer the reader to [1], [2], [3] for more details. Pairings and other parameters should be selected in proactive for efficiency and security. For appropriately selected parameters, the following assumptions should hold unconditionally.

### A. Assumptions

1) *Discrete Logarithm Assumption*: The *Discrete Logarithm Problem (DLP)* in $\mathbb{G}_1$ (and consequently $\mathbb{G}_2$) is intractable. In other words, given any two elements $P, Y \in \mathbb{G}_1$, computing $x \in \mathbb{Z}_q^*$ such that $Y = xP$ is hard.
2) *Diffie-Hellman Assumption*: Given $P, Y, R \in \mathbb{G}_1$ such that $Y = xP$ and $R = rY$ for unknowns $x, r \in \mathbb{Z}_q^*$, computing $U = rP$ is infeasible. This is the *Diffie-Hellman Problem (DHP)*.
3) *Extended Diffie-Hellman Assumption*: Given $P, Y, R \in \mathbb{G}_1$ such that $Y = xP$ and $R = rY$ for unknowns $x, r \in \mathbb{Z}_q^*$, computing $U = r^2P$ is infeasible. We call this the *Extended Diffie-Hellman Problem (EDHP)*.
4) *Extended Decisional Diffie-Hellman Assumption*: Given $P, Y, R, U \in \mathbb{G}_1$ such that $Y = xP$ and $R = rY$ for unknowns $x, r \in \mathbb{Z}_q^*$, deciding if $U = r^2P$ with probability $> 1/2$ is infeasible. This is the *Extended Decisional Diffie-Hellman Problem (EDDHP)*. (We note that the Decisional Diffie-Hellman Problem (DDHP), which requires deciding if $U = rP$ is easy in this case, since it only requires two pairing computations to decide if $e(P, R) = e(Y, U)$. All pairing based schemes make use of this observation [1]).
5) *Linear Diffie-Hellman Assumption*: Given $P, Y, R \in \mathbb{G}_1$ such that $Y = xP$ and $R = rY$ for unknowns $x, r \in \mathbb{Z}_q^*$, computing *any* pair $\langle Z, Q\rangle$ such that $Z = rP + xQ$ is infeasible. We call this the *Linear Diffie-Hellman Problem (LDHP)*.
6) *Linear Decisional Diffie-Hellman Assumption*: This relates to the decisional variant of the LDHP. Given $P, Y, R, Z, Q \in \mathbb{G}_1$ such that $Y = xP$ and $R = rY$ for unknowns $x, r \in \mathbb{Z}_q^*$, deciding if $Z = rP + xQ$ with probability $> 1/2$ is infeasible. We call this the *Linear Decisional Diffie-Hellman Problem (LDDHP)*.

The observed relationship between the different problems considered above is depicted in figure 1. An arrow indicates a reduction with the tail end representing the harder problem [4].

$$\begin{tikzcd}
DHP\ (Hard) \arrow[r, Leftrightarrow] & EDHP & EDDHP\ (Hard\ ?) \arrow[l, Rightarrow] \\
LDHP\ (Hard\ ?) \arrow[ur, Rightarrow] & & LDDHP\ (Hard\ ?) \arrow[ul, Rightarrow]
\end{tikzcd}$$

Fig. 1. Problem Hierarchy

## III. Setup PKI

In the rest of the discussion, we will using a PKI which will be setup as follows: A central authority is responsible for generating the security parameters. A trusted CA is responsible for certifying the public keys. To participate in the protocol each user must have a certified public key.

1) Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ be a bilinear mapping and $P \in \mathbb{G}_1$ be a generator of $\mathbb{G}_1$. The parameters $\langle e, \mathbb{G}_1, P\rangle$ are generated by the trusted authority and made public in an authentic way.
2) Each participant $\mathcal{ID}_i$ generates $x_i \leftarrow \mathbb{Z}_q$ as the private key. The corresponding public key is $Y_i = x_iP \in \mathbb{G}_1$. Each user also obtains a certificate from the CA linking the identity $\mathcal{ID}_i$ and the public key $Y_i$. In other words, the CA fixes the pairs $\langle Y_i, \mathcal{ID}_i\rangle$.

## IV. Blind Group Identification

This scheme enables a group of users to identify themselves to a server such that: (a) The identification test passes if none of the users cheat, (b) if any users cheat, the test will fail with a high probability, (c) it is not possible for the server or the users to know which person cheated. An important application for this type of scheme is in the following type of group systems: Assume that two users Alice and Bob want to identity themselves jointly to a server (for example, because they don't trust each other to individually login to the server without the other's approval). Alice wants to ensure that the identification succeeds if and only if the other user is really Bob. Bob has a similar requirement.

Assume that $\{\mathcal{ID}_1, \mathcal{ID}_2, \dots \mathcal{ID}_n\}$ are the set of users who want to jointly identify themselves. It is necessary that each user $\mathcal{ID}_i$ must have a certified public key $Y_i$ as described earlier. The goal of the protocol is that all users will simultaneously identify themselves to a server $\mathcal{S}$.

In other words, each user $\mathcal{ID}_i$ will prove possession of the discrete logarithm $x_i$ of $Y_i$ (to base P) such that $\mathcal{S}$ cannot be convinced about any of the individual statements separately. That is, the proof is valid only on all the statements together: "$\mathcal{ID}_i$ knows $x_i$" $\forall i : 1 \leq i \leq n$ but not on any of the individual statements like "$\mathcal{ID}_1$ knows $x_1$" or "$\mathcal{ID}_2$ knows $x_2$" independently of the others. We will assume the infrastructure of section III. The identification is done as follows:

1) The $n$ provers $\mathcal{ID}_1, \mathcal{ID}_2 \dots \mathcal{ID}_n$ start by claiming to $\mathcal{S}$ that they know the discrete logarithms $x_1, x_2, \dots x_n \in \mathbb{Z}_q$ of $Y_1, Y_2, \dots Y_n \in \mathbb{G}_1$ (to base $P$) respectively.
2) The verifier $\mathcal{S}$ generates $r_1, r_2, \dots r_n \leftarrow \mathbb{Z}_q$ and computes $R_i = r_i Y_i$ and $U_i = r_i^2 P$. It makes the list of challenges $\langle \mathcal{ID}_i, R_i, U_i \rangle$ public.
3) Each $\mathcal{ID}_i$ computes $V_i = \frac{1}{x_i} R_i$ and checks that $e(V_i, V_i) = e(U_i, P)$. If this test passes, it generates $Q_i \leftarrow \mathbb{G}_1$ and computes $Z_i = V_i + x_i Q_i$
4) All users then collaborate to jointly compute the value $Z = \sum_{j=1}^{j=n} Z_j$. This computation is hidden from $\mathcal{S}$ so that individual values $Z_j$ are effectively kept secret from it's view. The combined proof $\{Z, Q_1, Q_2 \dots Q_n\}$ is sent to $\mathcal{S}$.
5) $\mathcal{S}$ accepts if $e(Z - \sum_{j=1}^{j=n} r_j P, P) = \prod_{j=1}^{j=n} e(Q_j, Y_j)$.

We claim that this test will pass if and only if each $\mathcal{ID}_i$ knows $x_i$. To summarize the goals of the protocol, the individual users can jointly authenticate themselves to the server such that:

(a) If all users are honest, the server always accepts.
(b) If any of the users are dishonest, the server rejects with a high probability.
(c) The protocol is zero knowledge. It is not possible for anyone (including the server) to know which user cheated.
(d) Collusions are possible between users but not with the server (the server is trusted).

The protocol is secure based on the following analysis.

### A. *Correctness And Soundness*

The properties of bilinear maps ensure that the verification is always successful if none of $\mathcal{ID}_i$ cheat while the soundness property holds because:

1) Dishonest Prover: Computing individual proofs $\langle Z_i, Q_i \rangle$ without $x_i$ or $r_i$ is infeasible due to the linear Diffie-Hellman assumption. That is, a dishonest prover must solve the LDHP in order to cheat.
2) Dishonest Verifier: A dishonest verifier will generate $R$ non-randomly and will therefore not know $r_i$ corresponding to $R_i$. Due to this it will be hard for this verifier to generate $U_i$ such that $e(\frac{1}{x_i} R_i, \frac{1}{x_i} R_i) = e(U_i, P)$ due to the hardness of the EDHP. Thus a dishonest verifier will not be able to make anyone accept his/her challenge as valid as shown in [4]. We also note that the task of a dishonest verifier (EDHP) is considerably harder than that of a dishonest verifier (LDHP). This ensures an implicit (dishonest verifier) zero-knowledge property.

### B. *Zero Knowledge*

The case of a dishonest verifier can be ignored due to the above reasoning. We analyze the zero knowledge property for the following:

1) Honest Verifier Zero Knowledge: $\mathcal{S}$ can generate a valid accepting transcript on its own corresponding to $\{r_i, R_i, U_i\}$ $\forall i : 1 \leq i \leq n$ as follows: $\mathcal{S}$ generates $\alpha_i \leftarrow \mathbb{Z}_q$ $\forall i$ and computes $Q_i = \alpha_i P$, $R_i = r_i Y_i$. Then $Z = \sum_{j=1}^{j=n} r_i P + \alpha_i Y_i$.
2) Honest Verifier Secrecy: We require that it is impossible for a passive adversary to decide the honesty of the verifier. The reasoning for the single user case can be extended here. That is, given $P, x_i P, R_i, U_i$, deciding if $e(\frac{1}{x_i} R_i, \frac{1}{x_i} R_i) = e(U_i, P)$ is infeasible without knowledge of $r_i$ or $x_i$ due to the hardness of the EDDHP [4].
3) Honest Prover Secrecy: Assume that all the provers are honest and thus, $\mathcal{S}$ will eventually accept. We require that it is impossible for a passive adversary (including the provers) to decide the honesty some prover. We note that given $P, x_i P, r_i x_i P, r_i^2 P, Q_i, Z_i$, deciding if $Z_i = r_i P + x_i Q_i$ is infeasible without knowledge of $r$ or $x$ due to the hardness of the LDDHP in $\mathbb{G}_1$ as shown in [4]. Thus a passive adversary cannot decide the outcome of the identification.
4) Dishonest Prover Secrecy: Assume that some of the provers are dishonest. In this case, deciding if any given $Z_i = \frac{1}{x_i} R_i + x_i Q_i$ is infeasible without knowledge of $x_i$ or $r_i$ due to the decisional linear Diffie-Hellman assumption. That is, given $P, x_i P, r_i x_i P, Q, Z_i \in \mathbb{G}_1$, deciding if $Z_i = r_i P + x_i Q$ is infeasible without knowing at least one of $\{x_i, r_i\}$. Therefore if $\mathcal{S}$ rejects, none of the provers know which pairs $\langle Z_i, Q_i \rangle$ correspond to invalid proofs (if the individual coin tosses $r_i$ of $\mathcal{S}$ are kept secret and $\mathcal{S}$ is honest, no information is leaked to the provers). Similarly if the individual values $Z_i$ are kept secret (from $\mathcal{S}$), the identity of the dishonest provers is still concealed since computing individual proofs $Z_i$ just from $Z, Q_1, Q_2 \dots Q_n$ such that $Z_i = r_i P + x_i Q_i$ $\forall i$ is infeasible without knowledge of each $x_i$ due to the hardness of the DHP in $\mathbb{G}_1$ as shown in theorem 4.4 of [5] (cf. aggregate extraction). Consequently, even the verifier $\mathcal{S}$ does not have the ability to decide which of the provers are dishonest.

   Finally, if the joint computation of $Z$ is carried out in a way that any one individual prover or a small coalition of provers can know $Z_i$'s for only a small fraction of users, the identities of dishonest provers can still be effectively hidden, even if $\mathcal{S}$ can be coerced to reveal all the coin tosses $r_i$.

## V. Aggregate signatures

Aggregate signatures first introduced by Boneh et al. [5] allow a number of signatures on the same or different messages by different users to be verified in one step. The advantage of this scheme is evident in scenarios where batch authentication is possible. A very short aggregate signature scheme based on pairings is presented in [5]. Usually aggregate signatures can have the following variants:

1) Signatures of different users on different messages are verified in a batch.
2) Signatures of the same user on different messages are verified in a batch.
3) Signatures of different users on the same message are verified in a batch.

Efficient constructions for each of these variants can be constructed from the schemes of [5]. It should be noted that these variants are mutually exclusive. That is, in variant (1), it is necessary that each user be distinct and each message be distinct. We briefly describe the scheme here.[1] A set of $n$ users $\{\mathcal{ID}_1, \mathcal{ID}_2, \ldots \mathcal{ID}_n\}$ will create individual signatures on messages $\{m_1, m_2 \ldots m_n\}$ which will be verified in one step. We will assume the infrastructure of section III where the private key of $\mathcal{ID}_i$ is $x_i$ corresponding to the public key $Y_i = x_i P$ . The individual signature of $\mathcal{ID}_i$ is $s_i = x_i \mathcal{H}(m_i)$ while the aggregate signature is $s = \sum_{r=1}^{r=n} s_r$. The aggregate signature is accepted if $e(s, P) \overset{?}{=} \prod_{r=1}^{r=n} e(\mathcal{H}(m_r), Y_r)$. We refer the reader to [5] for the security proof.

## VI. Chained signatures

Another type of aggregate signatures, not considered in [5] are chained signatures. In this variant, an aggregation of signatures of different users on the same message is verified at once. Notice that there is a subtle change in the definition from aggregate signatures. Unlike aggregate signatures, the chained signatures have the following properties:

1) Any user can ‘add’ his signature to the aggregation of signatures, thereby adding his name to the list (or chain) of signers.
2) Once a user’s signature is added to the aggregation, it cannot be removed by any other user.
3) The order in which each user added to the aggregation is preserved (i.e. it cannot be changed)

Chained signatures allow any node to authenticate path of any received message and provide non-repudiation. The need for chained signatures is evident in scenarios where a number of participants are involved and the order of participants needs to be ascertained. Trivially, chained signatures can be realized by having each participant explicitly sign the identities of all previous participants along with the message and include it as part of the aggregate signature before passing it to the next participant. However, the scheme is efficient because the signature size increases linearly. We extend the aggregate signature scheme of [5] and construct very short constant-sized chained signatures. Some of the definitions are simplified since we do not provide a rigorous security proof in this paper (for which the reader is referred to [6]).

In this scenario there are $n$ ordered distinct participants $\langle \mathcal{ID}_1, \mathcal{ID}_2, \ldots \mathcal{ID}_n \rangle$ and one message $m$ to be signed. The original signer of the message $m$ is $\mathcal{ID}_1$. The message is passed from $\mathcal{ID}_i$ to $\mathcal{ID}_{i+1}$ along with a chain-signature as described below. We will use the infrastructure of section III.

### A. Signing

Let $L_r = \langle \mathcal{ID}_1, \mathcal{ID}_2, \ldots \mathcal{ID}_r \rangle$ for $r \geq 1$. Define $z_0 = 0 \in \mathbb{G}_1$. Define recursively $z_i = x_i \mathcal{H}(m, L_i) + z_{i-1}$. The chain-signature of $\mathcal{ID}_i$ on the message $m$ is $\langle z_i, L_i \rangle$

### B. Verification

$\mathcal{ID}_{i+1}$ accepts the signature $\langle z_i, L_i \rangle$ of $\mathcal{ID}_i$ as valid if $e(z_i, P) = \prod_{r=1}^{r=i} e(\mathcal{H}(m, L_r), Y_r)$.

### C. Non-Repudiation

Non-repudiation is provided as follows: using the signature $z_{i-1}$ received from $\mathcal{ID}_{i-1}$, user $\mathcal{ID}_i$ can prove to a judge that the message $m$ was indeed received from $\mathcal{ID}_{i-1}$ by invoking the verification procedure.

### D. Security Analysis

We will now give a brief analysis of the protocol and show that it achieves the necessary objectives of correctness and soundness. Roughly speaking, correctness requires that if all participants behave correctly, then the verification process should always output true. On the other hand, soundness requires that the verification process should output false with overwhelming probability if even one participant misbehaves. Recall that we want to ensure that the ordered list of participants specified in each $L_i$ should correctly and uniquely identify the path of the received message. The objectives of the protocol are summarized below:

(a) Using their private keys, participants can add their names to the end of the list (contained in the signature) without interaction. Arbitrary names cannot be added to a list without access to the corresponding private keys.
(b) Once added, a name cannot be removed from the list without access to the corresponding private key. However, the authenticity of the list can be verified (without interaction) using only the public keys of the participants involved.
(c) The authenticity of the list can be verified if and only if the correct order of all the participants is supplied.
(d) Non-repudiation must be provided. A holder of a chained signature should be able to convince a judge about the identities of all participants in the chain (in the correct order). The above properties automatically imply non-repudiation.

[1] It should be noted that the construction of aggregate signatures presented here is not ours and was presented in [5]. We include the scheme here for the sake of completeness

*1) 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: LHS of verification condition $= e(z_i, P) = e(\sum_{r=1}^{r=i} x_r \mathcal{H}(m, L_r), P) = \prod_{r=1}^{r=i} e(\mathcal{H}(m, L_r), x_r P)$ = RHS.

*2) Soundness:* To prove soundness of our scheme we need to show the verification process fails with a high probability if the protocol is not followed correctly. The standard accepted notion of soundness (i.e. *chosen ciphertext security*) for signatures is existential unforgeability [7], [8], [9]. To achieve chosen ciphertext security for chained signatures, however, two types of existential forgeries must be considered:

- Forgery on any previously unsigned messages
- Forgery on any previously unsigned sequence of identities

In other words, we must show that the signatures are unforgeable with respect to any chosen message $m$ or any chosen sequence $L_i$. This follows from the following theorem.

***Theorem 1**: Assume that the DHP in $\mathbb{G}_1$ is hard. Then our scheme is secure against existential forgery on messages or sequence of identities.*

***Proof***: We assume that the hash function $\mathcal{H}$ is a random oracle. It is easy to see that $z_i$ is an aggregation of individual signatures of $\mathcal{ID}_r$ on $\langle m, L_r\rangle \ \forall r \leq i$ using the aggregate signature scheme of [5]. To see this let $s_r \in \mathbb{G}_1$ be the individual signature of $\mathcal{ID}_r$ on $\langle m, L_r\rangle$ where $s_r = x_r\mathcal{H}(m, L_r)$. We can then rewrite:
$z_i = \sum_{r=1}^{r=i} x_r\mathcal{H}(m, L_r) = \sum_{r=1}^{r=i} x_r\mathcal{H}(m, L_r) = \sum_{r=1}^{r=i} s_r$

It is shown in theorem 3.2 of [1] that our scheme is secure against existential forgery of *individual* signatures if $\mathcal{H}$ is a random oracle. Assuming that $\mathcal{ID}_r \neq \mathcal{ID}_j$ whenever $r \neq j$, it is also ensured that $L_r \neq L_j$ whenever $r \neq j$ (in other words, all $L_r$ are distinct). It is shown in theorem 3.2 of [5] that this scheme is secure against existential forgery of *aggregate* signatures if the messages $\langle m, L_r\rangle$ are all distinct.

Consider any aggregation $z_i$ of individual signatures $\{s_1, s_2 \dots s_i\}$ such that none of the individual signatures are known. It is shown in theorem 4.5 of [5] that extracting any individual signature (or any sub-aggregation of individual signatures) from $z_i$ is not feasible if the DHP in $\mathbb{G}_1$ is hard.

To prove that our chained signature scheme is secure, it remains to be shown that an adversary cannot even change the order of the identities used in the verification. First note that existential forgery on individual signatures is not possible. If $\mathcal{H}$ is a random oracle, the probability of finding collisions of the type $\mathcal{H}(m, L_i) = \mathcal{H}(m', L_i')$ where $\langle m, L_i\rangle \neq \langle m', L_i'\rangle$ is negligible. This ensures that the pairs $\langle z_i, m\rangle$ and $\langle z_i, L_i\rangle$ are both fixed from the adversary's point of view. In other words, each individual signature $z_i$ corresponds to the unique sequence $L_i$ and a unique message $m \in \Sigma^*$. This completes the proof of security of our scheme. To summarize, this scheme is secure against the following attacks:

1) Existential forgery of aggregate signatures
2) Existential forgery of the order of signers
3) Extraction of individual signatures from an aggregation

### *E. Overview Of The Protocol*

We see that the chained signatures have an "additive" property, demonstrated by the fact that $\mathcal{ID}_{i+1}$ can 'add' more information to the signature $z_i$ of $\mathcal{ID}_i$ by computing $z_{i+1}$. A few points about this protocol are noteworthy:

1) Any user $\mathcal{ID}_i$ who passes the message can add its name in the list of the signature. Once added, users cannot remove names of other users from the list (without completely making the list empty), nor can they change the order or add new names.
2) 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.
3) The signature size is constant ignoring the payload of the identifier list (which cannot be avoided). The performance of the scheme can be summarized as follows (assuming $n$ users in the chain):
    a) Signing: one multiplication in $\mathbb{G}_1$, one addition in $\mathbb{G}_1$ and one computation of $\mathcal{H}$
    b) Verification: $n$ pairing computations and multiplications in $\mathbb{G}_2$, and $n$ computations of $\mathcal{H}$

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. It is easy to convert our backward chaining scheme to a forward chaining one simply by redefining $L_j$ in section as follows: $L_0 = \langle \mathcal{ID}_1\rangle$ and $L_k = \langle L_{k-1}, \mathcal{ID}_{k+1}\rangle$ if $k > 0$. We note that forward chaining schemes can be trivially made without pairings as described in [6].[2] The resulting signatures, however, are inefficient in size. We are not aware of any simple constructions for backward chaining schemes without pairings. In this regard, our scheme is unique because it enables ad-hoc backward chaining without involving any third parties.

[2] Denote by $\texttt{SIGN}_i$ the signing function of $\mathcal{ID}_i$ using any regular signature scheme. Let $\mathcal{H}$ be any hash function and let $m$ be the message to be signed. The ordered list of the first $i$ users is denoted as usual by the sequence $L_i = \langle \mathcal{ID}_1, \mathcal{ID}_2, \dots \mathcal{ID}_i\rangle$. We also define $f_j = \texttt{SIGN}_i(\mathcal{H}(m, L_{i+1}))$. The forward chain signature of $\mathcal{ID}_i$ in this scheme is the ordered $i+1$-tuple $\langle L_{i+1}, f_1, f_2, \dots f_i\rangle$ consisting of $i$ signatures and a sequence identifier. To verify the signature check that each $f_j$ is a valid signature of $\mathcal{ID}_j$ on $\mathcal{H}(m, L_{j+1}) \ \forall j \leq i$. This scheme is clearly inefficient because the signature size increases linearly as $\mathcal{O}(n)$

## VII. An Application Example: SPAM prevention

In this section, we consider possible applications of additive proofs. Specifically we focus only on chained signatures. A number of applications of chained signatures have been proposed in [10]. Here, we present a method to prevent spam (or unsolicited e-mail) using chained signatures. Our method essentially involves path authentication of any received mail.

Since it is impossible to completely stop spam, we propose a combination of proactive and reactive measures. Using the previous notation, the set of mail relays is $\{\mathcal{ID}_1, \mathcal{ID}_2, \ldots \mathcal{ID}_n\}$. We do not involve the senders or recipients simply because we feel that this process should be completely transparent to the end users. The only time when a recipient is involved is when an e-mail is to be reported as spam. Our approach is based on the following assumptions:

1) We first assume that spam (and all other mail) can be classified according to the path (of relays) it follows to reach a recipient. In other words, the path of any received mail can be accurately determined.
2) Due to assumption 1, the *first* relay mentioned in the path (of relays) for a spam mail is automatically considered responsible unless it is able to delegate this liability to a different relay.
3) A successful mail will be accepted for forwarding if and only if it is accompanied by a valid *backward chain signature* (as described in section VI keeping $m$ as the mail message).
4) The above assumption ensures that even if some relay accepts a message without a valid signature (a) either the message will be rejected by the next relay that validates the signature or (b) if this relay includes its own valid signature, it will automatically become liable for spam according assumption 2.
5) The use of a chain signature ensures that intermediate names in the list cannot be deleted unless *all* names are deleted. In this case, the relay who deletes the names will automatically become liable according to assumption 2.
6) Reactive measures (like blacklisting) can be taken against a relay continuously generating spam.
7) To ensure smooth integration to the existing email infrastructure, the sender of an email need not worry about the signing process. Only the relays would be responsible for the entire authentication process. It is the duty of each relay to sign only those emails originating from its local users. Otherwise, it will automatically become liable according to assumption 2.

We believe that this approach to classifying, enforcing and blacklisting relays using backward chain signatures will efficiently reduce spam to an acceptable level. The use of backward chaining ensures that the same message destined for multiple recipients (and having branching paths) need only be signed once at each node. We observe that the verification process involves many (computationally intensive) pairing computations. However, typically the number of relays involved for a mail delivery is very small (usually 2-3). This ensures that only a few pairing computations are carried out during verification.

## VIII. Conclusion

In this paper we described the notion of additive proofs. In a nutshell, an additive (zero-knowledge) proof is a composite proof constructed from (zero-knowledge) proofs of many individual statements. The intriguing property of such zero-knowledge proofs is that even though the composite proof itself proves the correctness of all the individual statements together, none of the individual statements can be proved independently of the others. Therefore, an inherent property of an additive zero-knowledge proof is that *the proof itself is a zero-knowledge proof of the fact that "it is a composite proof"*. As concrete examples of additive proofs, we presented three constructions: (a) blind group identification, (b) aggregate signatures and (c) backward chain signatures. Finally, as a useful application of backward chained signatures, we presented a novel method to prevent SPAM.

## References

[1] 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.

[2] 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.

[3] 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.

[4] Amitabh Saxena, Ben Soh, and Serguey Priymak. Zero-knowledge blind identification for smart cards using bilinear pairings. Cryptology ePrint Archive, Report 2005/343, 2005.

[5] 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.

[6] Amitabh Saxena and Ben Soh. A mobile agent authentication protocol using signature chaining with bilinear pairings. Cryptology ePrint Archive, Report 2005/272, 2005.

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

[8] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 433–444, London, UK, 1992. Springer-Verlag.

[9] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, New York, NY, USA, 1990. ACM Press.

[10] Amitabh Saxena and Ben Soh. One-way signature chaining: A new paradigm for group cryptosystems and e-commerce. Cryptology ePrint Archive, Report 2005/335, 2005.