A Special Purpose Signature Scheme for Secure Computation of Traffic in a Distributed Network

S. Sree Vivek*
TCS Lab, BSB 324
Dept. of Computer Science and Engineering
IIT Madras, Chennai, India. 600036
svivek@cse.iitm.ac.in

S. Sharmila Deva Selvi
TCS Lab, BSB 324
Dept. of Computer Science and Engineering
IIT Madras, Chennai, India. 600036
sharmila@cse.iitm.ac.in

Ramarathnam Venkatesan
Microsoft Research
One Microsoft Way
Redmond, Washington 98052, USA
venkie@microsoft.com

C. Pandu Rangan
TCS Lab, BSB 324
Dept. of Computer Science and Engineering
IIT Madras, Chennai, India. 600036
prangan@cse.iitm.ac.in

Abstract

We study the problem of traffic aggregation in a network with some natural security constraints. Here each node i has traffic (number of packets) m_i and they forward this information through the network to a server node which wishes compute $\sum m_i$. The sever node should not be able to know any additional information about m_i's and an intermediate node should not be able to tamper (without detection) the traffic information it is forwarding. We formalize this problem and suggest a solution using a variant of aggregate signatures, and prove its security using standard hardness assumptions.

Keywords: Aggregate signature scheme with message aggregation, Random oracle model, network traffic computation, secure computation of total traffic

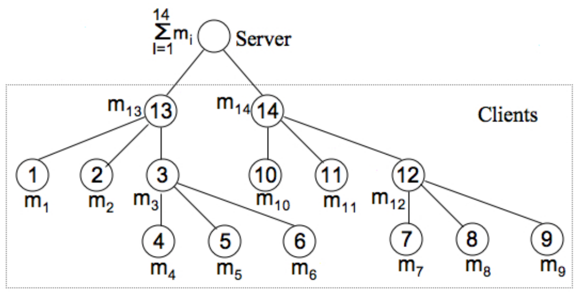
1 Introduction

Secure multi-party computation was initiated by Yao [1] by introducing the millionaire problem. Two millionaires A and B wants to find who is richer, without revealing the exact amount each has. Yao proposed a solution, for this problem, which paved way to a generalized notion called multi-party computation (MPC) protocols. In particular, a MPC protocol is used to compute the value of a public function F on N variables on points $(m_1, m_2, \ldots, m_N)$, by N participants $p_1, p_2, \ldots, p_N$ each having private data $m_1, m_2, \ldots, m_N$ respectively. An MPC protocol is secure if no participant learns more than the publicly known description of F and the result of $F(m_1, m_2, \ldots, m_N)$. In our work, $F(m_1, m_2, \ldots, m_N) = \sum m_i$, where the security properties are enforced with a variant of aggregate signature scheme.

In an *Aggregate Signature* scheme N participants $p_1, p_2, \ldots, p_N$ sign N messages say, $m_1, m_2, \ldots, m_N$ to generate a single signature. The original motivation was to reduce the costs of communication and computation during signature verification. The main desired security property of an aggregate signature is that a forger should neither be able to extract any individual signature from an aggregate signature nor be able to generate an aggregate signature without the private keys of all the N participants. Several aggregate signatures with diverse properties were proposed in the literature [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

In computer networking, *Network Traffic Control* is an important task. To manage the network, the administrator should know the total traffic (number of packets) in the various parts of the network.

*Corresponding author: Tel: +91-0091-4422575387, Email: svivek@cse.iitm.ac.in



Constraints:

- Server and clients should not know the individual traffic of any other node of the system.
- No forger should be able to alter the individual traffic value or the total traffic computed by the server.
- None except the server finds the total traffic in the system.

Figure 1: An example scenario

This allows one to detect packet losses and congestion. The individual node may forward the traffic information to its neighbors and a malicious or faulty node may corrupt the traffic information. We focus on a hierarchal traffic aggregation on a tree which is embedded on an arbitrary network.

Each client node i generates a traffic. Each leaf node simply forwards its m_i's to its parents. Each client node i which is a non-leaf node computes the sum of all incoming traffic from its children along with its own traffic m_i and forwards the sum to its parent. The server is located at the root and would like to compute the total traffic ($\sum m_i$) in the network. This is shown in figure 1. The two security requirements are:

1. The server should not be able to compute any information about the traffic of individual clients.
2. A client i should not be able to tamper with the traffic forwarded to it.

We assume the total traffic $\sum m_i$ is bound by a polynomial. We formalize this problem and provide a solution using standard hardness assumptions. The computational load on the server is $O(\sqrt{\sum m_i})$, which is around 2^{15} field operations for a traffic of 1 billion packets and each node computes one signature. The two requirements regarding the computation of the traffic are:

1. The server should be able to compute the total traffic in the network.
2. No one else can compute the total traffic.

To achieve this property, we make use of the concept of aggregate signatures. The signature of each node gets aggregated with the signatures of its ancestor and hence the total traffic of all children nodes are summed up with the traffic of the parent node when the signature aggregation is done by the parent node. It should be noted that all existing aggregate signature schemes require the messages to be sent along with the signature to get verified by the verifier. Moreover, existing schemes do not allow secure computation on the messages that are signed. For our purpose, we require that the messages (namely the individual traffic in each node) should be summed up each time the signature is aggregated in a secure way, which make all existing schemes handicapped in achieving these properties.

Our Contribution: In this paper, we provide a formal security model for aggregate signatures with message aggregation, where the total traffic is polynomially bounded and provide a concrete scheme. We observe that, the scheme should provide individual signature unforgeability, aggregate signature unforgeability and confidentiality as the basic security requirements. We prove the security of our scheme in the random oracle model. In our construction, each parent node gets convinced that the total traffic which it received from its successors is not altered by any one during transit. Moreover, it can be easily observed that, if an individual node wants to quote a higher amount of traffic, it can always quote it and sign the hiked value but this cannot be restricted by any cryptographic protocol. Hence, we make the reasonable assumption that the nodes are ***honest*** in quoting their individual traffic but are ***curious*** in knowing the total traffic flowing through the network, and the individual traffic of other nodes.

2 Preliminaries

In this section, we review a few preliminary assumptions that are used to prove the scheme.

2.1 Bilinear Pairing

Let $\mathbb{G}_1$ be an additive cyclic group generated by P, with prime order q, and $\mathbb{G}_2$ be a multiplicative cyclic group of the same order q. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties.

- ***Bilinearity.*** For all $P,Q,R \in \mathbb{G}_1$,
  - $\hat{e}(P+Q,R) = \hat{e}(P,R)\hat{e}(Q,R)$
  - $\hat{e}(P,Q+R) = \hat{e}(P,Q)\hat{e}(P,R)$
  - $\hat{e}(aP,bQ) = \hat{e}(P,Q)^{ab}$ [Where $a,b \in_R \mathbb{Z}_q^*$]
- ***Non-Degeneracy.*** There exist $P,Q \in \mathbb{G}_1$ such that $\hat{e}(P,Q) \neq I_{\mathbb{G}_2}$, where $I_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
- ***Computability.*** There exists an efficient algorithm to compute $\hat{e}(P,Q)$ for all $P,Q \in \mathbb{G}_1$.

2.2 Computational Assumptions

In this section, we review the computational assumptions related to bilinear maps that are relevant to the protocol we discuss.

2.2.1 Computation Diffie-Hellman Problem (CDHP)

Definition 2.1. *Given $(P,aP,bP) \in \mathbb{G}_1^3$ for unknown $a,b \in \mathbb{Z}_q^*$, the CDH problem in $\mathbb{G}_1$ is to compute abP.*

Definition. *The advantage of any probabilistic polynomial time algorithm $\mathscr{A}$ in solving the CDH problem in $\mathbb{G}_1$ is defined as*

$$Adv_{\mathscr{A}}^{CDH} = Pr\left[\mathscr{A}(P,aP,bP) = abP \mid a,b \in \mathbb{Z}_q^*\right]$$

The CDH Assumption is that, for any probabilistic polynomial time algorithm $\mathscr{A}$, the advantage $Adv_{\mathscr{A}}^{CDH}$ is negligibly small.

3 Framework and Security Models

In this section, we propose the generic frame work for aggregate signature with message aggregation, frame the security goals and security model for the same. There are three entities in the scheme. i) A trusted Authority (TA) who is responsible for generating the private keys of the server and nodes of the system. ii) The server who is interested in calculating the sum of all messages signed by the client nodes in such a way that the server does not know the individual message sent by each node. iii) A network of client nodes arranged in a tree like fashion. Each client node verifies and aggregates the signature sent by its children and passes on to its parent node.

3.1 Generic Framework for Aggregate Signature with Message Aggregation

- **Setup(κ):** The trusted authority (TA) generates the public parameters *params* and makes them public.
- **Keygen:** The key generation algorithm is executed by the trusted authority to generate the private key of the server and the client nodes:
  - *N_i:* For each node N_i, TA computes the private key d_i and the public key is Y_i and sends them to the client node.
  - *Server:* The private key S of the server is computed by the TA and is passed on to the server. It should be noted that there should be a link between the private key of the server and the private keys of the client nodes.
- **Sign(m_i, d_i, w_j, k):** The input parameters for this algorithm are the message m_i which is polynomial in κ, the private key of the i^{th} node d_i, the session identifier of the j^{th} session w_j and the k^{th} execution of the sign algorithm in the session w_j is represented by k. The output of this algorithm is σ_i, the individual signature of the node N_i.
- **AggSign(σ_l, for $N_l \in \{Child(N_i)\}, m_i, d_i, w_j, k$):** The input parameters for this algorithm are σ_l, for $N_l \in \{Child(N_i)\}$, the signatures aggregated by $N_l \in \{Child(N_i)\}$ the message m_i which is polynomial in κ, d_i the private key of the i^{th} node, the session identifier of the j^{th} session w_j and the k^{th} execution of the sign algorithm in the session w_j is represented by k. The output of this algorithm is σ_i, which is the aggregation of all the signatures σ_l, for $N_l \in \{Child(N_i)\}$ with N_i's signature, when executed by the client nodes and when executed by the server it produces σ_{Agg}.
- **MessageRecover(σ_{Agg}, S, w_j, k):** The input to this algorithm are the aggregate signature σ_{Agg}, the private key S of the server and $w_j$is the session identifier and k is the signature number in session w_j. The output of this algorithm is the summation of all the messages $\sum m_i$.

3.2 Security Goals

First, we set the security goals for the aggregate signature scheme with message aggregation.

- In the scheme, all client nodes are semi-trusted in the sense that any node cannot alter the message signed by another node but can maliciously participate in the protocol, i.e. can sign on a message that may attempt to increase or decrease the overall sum of messages without being identified by the server.
- **Individual Signature Unforgeability**: The individual signatures of the client nodes should be unforgeable with respect to existential forgery by any entity except the TA of the system.
- **Aggregate Signature Unforgeability**: The aggregate signature should be unforgeable with respect to existential forgery and even though the server has some information about the user secret keys, it should not be able to forge a valid aggregate signature.
- **Confidentiality**: The server or any client node should not be able to distinguish the message signed by any other client node in the system.
- **Completeness**: The server should be convinced that all the client nodes have participated in the transaction, i.e. even if one of the client nodes have not contributed in the signature generation, the server should not be able to retrieve the aggregated message.

3.3 Security Model

In this section, we provide the security model for proving the Individual Signature Unforgeability, Aggregate Signature Unforgeability, Confidentiality and Completeness of the proposed aggregate signature with message aggregation.

3.3.1 I - Individual Signature Unforgeability:

An individual signature is claimed to be existentially unforgeable under chosen message attack (EUF-CMA), if any polynomially bounded forger $\mathscr{F}$ has a negligible advantage in the following game:

We consider the scenario wherein the client nodes are arranged in a hierarchy, so it forms a tree structure. Thus, the response to *Sign* and *AggSign* will be made by $\mathscr{C}$ with respect to the position of the current node. The queries with respect to the *MessageRecover* oracle will be responded by $\mathscr{C}$ only if all nodes have contributed in the generation of the aggregate signature.

Setup: Let us consider there are n client nodes in the system. The challenger $\mathscr{C}$ runs the **Setup** algorithm to generate the master public parameters *params*. $\mathscr{C}$ generates the private keys d_i (for $i = 1$ to $n-1$) and public key Y_i, (for $i = 1$ to n) and the private key S of the server by invoking the **Keygen** algorithm. $\mathscr{C}$ now gives *params*, d_i, (for $i = 1$ to $n-1$), Y_i, (for $i = 1$ to n) and S to $\mathscr{F}$.

Training Phase: $\mathscr{F}$ asks polynomial number of queries to the Sign, AggSign and MessageRecover oracles provided by $\mathscr{C}$ which are described below.

- *Sign oracle:* $\mathscr{F}$ can query the individual signature by any node for any session identifier w_j and signature number in the session k of a user with public key Y_i. $\mathscr{C}$ chooses a message m_i (of size polynomial in κ), computes $\sigma_i^{(1)}$ and $\sigma_i^{(2)}$. $\mathscr{C}$ sends $\sigma_i^{(2)}$ as the sign by the user with public key Y_i on $\sigma_i^{(2)}$. The query may also be related to the target node with public key Y_n.

- *AggSign oracle:* $\mathscr{F}$ can query the aggregate signature for any session identifier w_j and signature number in the session k to this oracle. $\mathscr{C}$ chooses message m_i (of size polynomial in κ), for $i = 1$ to n, aggregates them and sends the aggregate signature to $\mathscr{F}$. $\mathscr{F}$ verifies the validity of the aggregate signature and should be able to recover the aggregated message since $\mathscr{F}$ knows the private key of the server.

- *MessageRecover oracle:* $\mathscr{F}$ produces an aggregate signature σ_{Agg} and queries the corresponding aggregate message $\sum m_i$, (for $i = 1$ to n). $\mathscr{C}$ verifies the validity of the aggregate signature and should be able to recover the aggregated message since $\mathscr{C}$ knows the private key of the server.

Forgery: At the end of the ***Training Phase***, $\mathscr{F}$ produces a valid message m_i, signature σ^* pair. $\mathscr{F}$ wins the game if σ^* is a valid signature and σ^* is not the output of any previous queries to the *Sign Oracle* with m_i as the message, during the **Training Phase**.

3.3.2 II - Aggregate Signature Unforgeability:

An aggregate signature is claimed to be existentially unforgeable under chosen message attack (EUF-CMA), if any polynomially bounded forger $\mathscr{F}$ has a negligible advantage in the following game:

Setup: Let us consider there are n client nodes in the system. The challenger $\mathscr{C}$ runs the **Setup** algorithm to generate the master public parameters *params*. $\mathscr{C}$ generates the private keys d_i for $i = 1$ to $n-1$), the public key Y_i, (for $i = 1$ to n) and the private key S of the server by invoking the **Keygen** algorithm. $\mathscr{C}$ now gives *params*, d_i, (for $i = 1$ to $n-1$), Y_i, (for $i = 1$ to n) and S to $\mathscr{F}$.

Training Phase: $\mathscr{F}$ asks polynomial number of queries to the Sign, AggSign and MessageRecover oracles provided by $\mathscr{C}$ which are described below.

- *Sign oracle:* $\mathscr{F}$ can query the individual signature by any node for any session identifier w_j and signature number in the session k of a user with public key Y_i. $\mathscr{C}$ chooses a message m_i (of size polynomial in κ), computes $\sigma_i^{(1)}$ and $\sigma_i^{(2)}$. $\mathscr{C}$ sends $\sigma_i^{(2)}$ as the sign by the user with public key Y_i on $\sigma_i^{(2)}$. The query may also be related to the target node with public key Y_n.
- *AggSign oracle:* $\mathscr{F}$ can query the aggregate signature for any session identifier w_j and signature number in the session k to this oracle. $\mathscr{C}$ chooses message m_i (of size polynomial in κ), for $i = 1$ to n, aggregates them and sends the aggregate signature to $\mathscr{F}$. $\mathscr{F}$ verifies the validity of the aggregate signature and should be able to recover the aggregated message since $\mathscr{F}$ knows the private key of the server.
- *MessageRecover oracle:* $\mathscr{F}$ produces an aggregate signature σ_{Agg} and queries the corresponding aggregate message $\sum m_i$, (for $i = 1$ to n). $\mathscr{C}$ verifies the validity of the aggregate signature and should be able to recover the aggregated message since $\mathscr{C}$ knows the private key of the server.

Forgery: At the end of the ***Training Phase***, $\mathscr{F}$ produces a valid aggregate signature σ^*_{Agg}. $\mathscr{F}$ wins the game if σ^*_{Agg} is a valid signature and σ^*_{Agg} is not the output of any previous queries to the *AggSign Oracle* with the same session identifier or the signature number in the session as in the forged aggregate signature, during the **Training Phase**. It should be noted that $\mathscr{F}$ can produce the forgery for a message set which was already queried to the *AggSign Oracle* but for a different session identifier or the signature number in the session.

3.3.3 III - Confidentiality:

As the message is polynomial with respect to the security parameter κ, the usual way of proving confidentiality, namely CCA2 game (Adaptive Chosen Ciphertext Attack) cannot be used in this context. This is because, the adversary will always have a non-negligible advantage in winning the game by guessing the message. To be specific the advantage of the adversary will be $\frac{1}{\mathtt{Poly}(\kappa)}$. Thus, we advocate for *perfect secrecy* in this context.

Definition 3.1. *A cryptosystem offers Perfect Secrecy if* $Pr[m] = Pr[m|\sigma]$ *for all* $m \in \mathbb{M}$ *and* $\sigma \in \mathbb{C}$, *where* $\mathbb{M}$ *and* $\mathbb{C}$ *are the message space and the ciphertext space respectively. That is, the probability that the message is* m *is identical to the probability of* m *being the message even after the ciphertext* σ *is observed.*

4 Aggregate Signature with Message Aggregation (AG_MA):

In this section, we propose the new Aggregate Signature supporting Message Aggregation AG_MA and prove its security in the security model provided in section 3. Important notations used in the scheme:

$\mathbb{N}$ - Set of all client nodes.
N_i - Client node with index i.
$\mathbb{I}_i$ - Set of all indices of child nodes of node N_i.

4.1 The Scheme

Setup: The trusted authority (TA) chooses two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q, a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, chooses a random generator $P \in_R \mathbb{G}_1$, chooses two hash functions $F : \mathbb{G}_1 \rightarrow \mathbb{G}_1$ and $H : \{0,1\}^* \rightarrow \mathbb{G}_1$. The TA publishes the public parameters $params = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, H(.), F(.))$

Keygen: The key generation algorithm is executed by the trusted authority as follows:

- *N_i:* For each node $N_i \in \mathbb{N}$, TA chooses $x_i, y_i \in_R \mathbb{Z}_q^*$ and computes $Y_i = y_i P$. The private key of the node N_i is $d_i = \langle x_i, y_i \rangle$ and the public key is Y_i.

- *Server:* The private key of the server $S = \sum x_i P$, where $i = 1$ to $|\mathbb{N}|$. The server pre-computes the value $\beta = \hat{e}(P,P)$ and keeps it for the message recovery phase.

Sign(m_i, d_i, w_j, k): The input parameters for this algorithm are the message m_i which is polynomial in κ (Represented as `Poly`(κ)), the private key of the i^{th} node d_i, the session identifier of the j^{th} session w_j and the k^{th} execution of the Sign algorithm in the session w_j is represented by k. σ_i, the individual signature of the node N_i is computes as follows.

- Compute $H_{jk} = H(w_j, k)$.

- Compute the signature as $\sigma_i^{(1)} = x_i H_{jk} + m_i P$.

- Compute $F_i = F(\sigma_i^{(1)})$ and $\sigma_i^{(2)} = y_i F_i$.

- The signature on message m_i by node N_i is $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$.

Remark: Note that even if the size of m_i is polynomially bounded, the size of x_i is very large because $x_i \in \mathbb{Z}_q^*$ and hence any adversary cannot obtain m_i from $\sigma_i^{(1)}$.

AggSign($\langle \{\sigma_l\}, \forall l \in \mathbb{I}_i \rangle, m_i, d_i, w_j, k$): Let m_i be the message to be aggregated, d_i be the private key of the i^{th} node and w_j be the session identifier of the j^{th} session and k represent the k^{th} execution of the AggSign algorithm in the session w_j.

- The following steps are executed to obtain the aggregate signature when this algorithm is run by the server:

  - Computes $H_{jk} = H(w_j, k)$.
  - Checks for all nodes N_l, where $l \in \mathbb{I}_{server}$ whether, $\hat{e}(\sigma_l^{(2)}, P) \stackrel{?}{=} \hat{e}(F(\sigma_l^{(1)}), Y_i)$.
  - Computes $\sigma_{Agg} = \left(\sum_{l \in \mathbb{I}_{server}} (\sigma_l^{(1)}) \right)$.

- When this algorithm is executed by the i^{th} client node, the signature is aggregated as follows:

  - Computes $H_{jk} = H(w_j, k)$.
  - Checks for all nodes N_l, where $l \in \mathbb{I}_i$ whether, $\hat{e}(\sigma_l^{(2)}, P) \stackrel{?}{=} \hat{e}(F(\sigma_l^{(1)}), Y_i)$.
  - If all the above checks are valid, N_i computes the aggregate signature as

$$\sigma_i^{(1)} = \left(\sum_{l \in \mathbb{I}_i} (\sigma_l^{(1)}) \right) + x_i H_{jk} + m_i P$$

  - Computes $F_i = F(\sigma_i^{(1)})$ and $\sigma_i^{(2)} = y_i F_i$.
  - The signature $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$ on message m_i by node N_i is passed on to its parent node.

Remark: Note that the function of $\sigma_i^{(1)}$ is to aggregate the individual traffic (i.e. m_i) of the node with the sum of all traffics of its successors. This signature can easily be altered by an adversary, by just adding some random $m'P$ (where m' is polynomial in κ) to $\sigma_i^{(1)}$ but the adversary will not be able to generate $\sigma_i^{(2)}$ corresponding to the altered $\sigma_i^{(1)}$ value. Thus, $\sigma_i^{(2)}$ acts as a signature on $\sigma_i^{(1)}$ by the node that has generated $\sigma_i^{(1)}$ and $\sigma_i^{(1)}$ acts as the value that aggregates the individual traffic of the node and the total traffic of the successors of the node.

MessageRecover(σ_{Agg}, S, w_j, k): The server obtains σ_{Agg} by executing *AggSign* algorithm and recovers $\sum m_i$ as follows:

- Computes $H_{jk} = H(w_j, k)$.
- Computes $\alpha = \hat{e}(\sigma_{Agg}, P)\hat{e}(H_{jk}, S)^{-1}$
- It is easy to see that $\alpha = \beta^{\sum m_i}$.
- Now, the server solves DL of α with respect to β to find $\sum m_i$.

Remark: Note that since we made the assumption that m_i is polynomially bounded, $\sum m_i$ is also bound polynomially and hence small. Thus, Discrete Logarithm can be solved efficiently in reasonable time for this case. It should be further noted that this is a valid assumption because the value of a typical network traffic can only be polynomial with respect to the security parameter κ.

5 Security Proof

In this section we formally prove the security of our scheme.

5.1 Individual Signature Unforgeability

Theorem 5.1. *If there exists an algorithm to break the EUF-CMA security of the individual signature of the AG_MA scheme then there exists an algorithm $\mathscr{C}$ that can solve the CDH problem with the same advantage.*

Proof: The challenger $\mathscr{C}$ is challenged with an instance of the CDH problem say, $(P, aP, bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_q^*$. Let us consider that there exists a forger $\mathscr{F}$, who is capable of forging the individual signature of the AG_MA scheme, $\mathscr{C}$ can make use of $\mathscr{F}$ to compute abP by playing the following interactive game with $\mathscr{F}$.

Setup: Let us consider there are n client nodes in the system. The challenger $\mathscr{C}$ performs the following to setup the system:

- Chooses $x_i, y_i \in_R \mathbb{Z}_q^*$, (for $i = 1$ to $n-1$) and generates the private keys $d_i = \langle x_i, y_i \rangle$ and public keys $Y_i = y_i P$, (for $i = 1$ to $n-1$).
- Chooses $x_n \in_R \mathbb{Z}_q^*$, sets $Y_n = aP$ (here, aP is taken from the CDH instance) and the private key of the server as $S = \sum_{i=1}^{n-1} x_i P$.
- Designs the two hash functions H and F as random oracles $\mathscr{O}_H$ and $\mathscr{O}_F$. $\mathscr{C}$ maintains lists L_H and L_F in order to consistently respond to the queries to the random oracles $\mathscr{O}_H$ and $\mathscr{O}_F$.

$\mathscr{C}$ now gives $params = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P)$, y_i, (for $i = 1$ to $n-1$), x_i, (for $i = 1$ to n), Y_i, (for $i = 1$ to n) and S to $\mathscr{F}$.

Training Phase: $\mathscr{F}$ asks polynomial number of queries to the Sign, AggSign and MessageRecover oracles provided by $\mathscr{C}$. The design of the oracles and responses given by $\mathscr{C}$ to $\mathscr{F}$ are described below.

$\mathscr{O}_H(w_j, k)$**:** In order to respond to this query, $\mathscr{C}$ checks whether a tuple of the form $\langle w_j, k, r_{jk}, H_{jk} \rangle$ exists in the list L_H. If it exists, $\mathscr{C}$ returns H_{jk} as the response to this query, else $\mathscr{C}$ performs the following:

- Chooses $r_{jk} \in_R \mathbb{Z}_q^*$.
- Computes the value $H_{jk} = r_{jk}P$.
- Stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Returns H_{jk} as the response to this query.

$\mathscr{O}_F(\sigma_i^{(1)})$**:** When this oracle is queried with input $\sigma^{(1)}$, $\mathscr{C}$ checks whether a tuple of the form $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ exists in the list L_F. If it exists, $\mathscr{C}$ returns the corresponding F_i to $\mathscr{F}$, else performs the following to answer this query:

- Chooses $f_i \in_R \mathbb{Z}_q^*$.
- Computes the value $F_i = f_i bP$ (Where bP is taken from the CDH instance).
- Stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F.
- Returns F_i as the response to this oracle.

$\mathscr{O}_{Sign}(w_j, k)$**:** If the signature does not correspond to the target node N_n, $\mathscr{C}$ proceeds as per the sign algorithm, otherwise $\mathscr{C}$ responds as follows:

- Chooses $r_{jk}, f_i \in_R \mathbb{Z}_q^*$ and a message $m_i \in_R \texttt{Poly}(\kappa)$.
- Computes the value $H_{jk} = r_{jk}P$.
- Stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Computes the signature $\sigma_i^{(1)} = x_i H_{jk} + m_i P$.
- Computes the value $F_i = f_i P$.
- Stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F.
- Computes $\sigma_i^{(2)} = f_i(bP)$. (Here, bP is taken from the CDH problem.)
- Returns the signature $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$ to $\mathscr{F}$.

$\mathscr{O}_{AggSign}(\langle \{\sigma_l\}, \forall l \in \mathbb{I}_i \rangle, m_i, w_j, k)$**:** $\mathscr{F}$ can query the aggregate signature on any set of signatures $\langle \{\sigma_l\}, \forall l \in \mathbb{I}_i \rangle$. $\mathscr{C}$ responds as follows:

if N_i is the root node (essentially, the server), $\mathscr{C}$ performs the following:

- Computes $\sigma_{Agg} = \left(\sum_{l \in \mathbb{I}_{server}} (\sigma_l^{(1)}) \right)$ and sends σ_{Agg} as the response to $\mathscr{F}$.

if N_i is not the root node (essentially, not the server), $\mathscr{C}$ performs the following:

- Chooses $r_{jk}, f_i \in_R \mathbb{Z}_q^*$.
- Sets $H_{jk} = r_{jk}P$ and stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Checks for all nodes N_l, where $l \in \mathbb{I}_i$ whether, $\hat{e}(\sigma_l^{(2)}, P) \stackrel{?}{=} \hat{e}(F(\sigma_l^{(1)}), Y_i)$.
- If all the above checks are valid, $\mathscr{C}$ computes the aggregate signature as $\sigma_i^{(1)} = \left(\sum_{l \in \mathbb{I}_i} (\sigma_l^{(1)}) \right) + x_i H_{jk} + m_i P$.
- Computes the signature on the aggregate signature as follows:
  - If $i = n$, computes the value $F_i = f_i P$, stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F and computes $\sigma_i^{(2)} = f_i(bP)$. (Here, bP is taken from the CDH problem.)
  - If $i \neq n$, computes the value $F_i = f_i P$, stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F and computes $\sigma_i^{(2)} = y_i(f_i P)$. (Since $\mathscr{C}$ knows the private key corresponding to the nodes N_i, for $i = 1$ to $n-1$).
- The signature $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$ is returned to $\mathscr{F}$ a the aggregate signature by node N_i.

$\mathscr{O}_{MessageRecover}(\sigma_{Agg}, S, w_j, k)$**:** In order to respond to this query $\mathscr{C}$ performs the following:

- $\mathscr{C}$ checks for a tuple of the form $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H and retrieves the corresponding H_{jk}.
- Since $\mathscr{C}$ knows $\sum_{i=1}^{n} x_i$, computes $\sum_{i=1}^{n} x_i(H_{jk})$.
- Computes $\sum m_i P = \sigma_{Agg} - \sum_{i=1}^{n} x_i(H_{jk})$.
- Solves DL on $\sum m_i P$ with respect to P, retrieves $\sum m_i$ and returns it to $\mathscr{F}$

Forgery: At the end of the ***Training Phase***, $\mathscr{F}$ produces a valid signature $\sigma_n^* = \langle \sigma_n^{*(1)}, \sigma_n^{*(2)} \rangle$. $\mathscr{C}$ retrieves the solution for the CDH problem from it as follows:

- $\mathscr{C}$ checks for the tuple of the form $\langle \sigma_i^{*(1)}, f_i, F_i \rangle$ in the list L_F and retrieves f_i. ($\mathscr{C}$ knows that $\sigma_n^{*(2)} = y_n F_i$.)
- Since, F_i was set to be $f_i bP$ during $\mathscr{O}_F$ query corresponding to $\sigma_i^{*(1)}$ and $Y_n = y_n P$ is set to be aP during the setup, $\mathscr{C}$ can compute $abP = f_i^{-1} \sigma_n^{*(2)}$.

5.2 Aggregate Signature Unforgeability

Theorem 5.2. *If there exists an algorithm to existentially forge the aggregate signature of the AG_MA scheme then there exists an algorithm $\mathscr{C}$ that can solve the CDH problem with the same advantage.*

Proof: The challenger $\mathscr{C}$ is challenged with an instance of the CDH problem say, $(P, aP, bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_q^*$. Let us consider that there exists a forger $\mathscr{F}$, who is capable of forging the aggregate signature of the AG_MA scheme, $\mathscr{C}$ can make use of $\mathscr{F}$ to compute abP by playing the following interactive game with $\mathscr{F}$.

Setup: Let us consider there are n client nodes in the system. The challenger $\mathscr{C}$ performs the following to setup the system:

- Chooses $x_i, y_i \in_R \mathbb{Z}_q^*$, (for $i = 1$ to $n-1$) and generates the private keys $d_i = \langle x_i, y_i \rangle$ and public keys $Y_i = y_i P$, (for $i = 1$ to $n-1$).
- Chooses $y_n \in_R \mathbb{Z}_q^*$, sets $Y_n = y_n P$ and the private key of the server as $S = \sum_{i=1}^{n-1} x_i P + aP$ (here, aP is taken from the CDH instance) .
- Designs the two hash functions H and F as random oracles $\mathcal{O}_H$ and $\mathcal{O}_F$. $\mathcal{C}$ maintains lists L_H and L_F in order to consistently respond to the queries to the random oracles $\mathcal{O}_H$ and $\mathcal{O}_F$.

$\mathcal{C}$ now gives $params = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P)$, y_i, (for $i = 1$ to n), x_i, (for $i = 1$ to $n-1$), Y_i, (for $i = 1$ to n) and S to $\mathcal{F}$.

Training Phase: $\mathcal{F}$ asks polynomial number of queries to the Sign, AggSign and MessageRecover oracles provided by $\mathcal{C}$. The design of the oracles and responses given by $\mathcal{C}$ to $\mathcal{F}$ are described below.

$\mathcal{O}_H(w_j, k)$**:** In order to respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle w_j, k, r_{jk}, H_{jk} \rangle$ exists in the list L_H. If it exists, $\mathcal{C}$ returns H_{jk} as the response to this query, else $\mathcal{C}$ performs the following:

- Chooses $r_{jk} \in_R \mathbb{Z}_q^*$.
- Computes the value $H_{jk} = r_{jk} bP$ (here, bP is taken from the CDH instance) .
- Stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Returns H_{jk} as the response to this query.

$\mathcal{O}_F(\sigma_i^{(1)})$**:** When this oracle is queried with input $\sigma^{(1)}$, $\mathcal{C}$ checks whether a tuple of the form $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ exists in the list L_F. If it exists, $\mathcal{C}$ returns the corresponding F_i to $\mathcal{F}$, else performs the following to answer this query:

- Chooses $f_i \in_R \mathbb{Z}_q^*$.
- Computes the value $F_i = f_i P$.
- Stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F.
- Returns F_i as the response to this oracle.

$\mathcal{O}_{Sign}(w_j, k)$**:** If the signature does not correspond to the target node N_n, $\mathcal{C}$ proceeds as per the sign algorithm, otherwise $\mathcal{C}$ responds as follows:

- Chooses $r_{jk}, f_i \in_R \mathbb{Z}_q^*$ and a message $m_i \in_R \texttt{Poly}(\kappa)$.
- Computes the value $H_{jk} = r_{jk} P$.
- Stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Computes the signature $\sigma_i^{(1)} = r_{jk} aP + m_i P$.
- Computes the value $F_i = f_i P$.
- Stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F.
- Computes $\sigma_i^{(2)} = f_i P$.

- Returns the signature $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$ to $\mathscr{F}$.

$\mathscr{O}_{AggSign}(\langle \{\sigma_l\}, \forall l \in \mathbb{I}_i \rangle, m_i, w_j, k)$**:** $\mathscr{F}$ can query the aggregate signature on any set of signatures $\langle \{\sigma_l\}, \forall l \in \mathbb{I}_i \rangle$. $\mathscr{C}$ responds as follows:
if N_i is the root node (essentially, the server), $\mathscr{C}$ performs the following:

- Computes $\sigma_{Agg} = \left(\sum_{l \in \mathbb{I}_{server}} (\sigma_l^{(1)}) \right)$ and sends σ_{Agg} as the response to $\mathscr{F}$.

if N_i is not the root node (essentially, not the server), $\mathscr{C}$ performs the following:

- Chooses $r_{jk}, f_i \in_R \mathbb{Z}_q^*$.
- Sets $H_{jk} = r_{jk}P$ and stores the tuple $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H.
- Checks for all nodes N_l, where $l \in \mathbb{I}_i$ whether, $\hat{e}(\sigma_l^{(2)}, P) \overset{?}{=} \hat{e}(F(\sigma_l^{(1)}), Y_i)$.
- If all the above checks are valid, $\mathscr{C}$ computes the aggregate signature as follows:
  - If $i = n$, computes $\sigma_i^{(1)} = \left(\sum_{l \in \mathbb{I}_i} (\sigma_l^{(1)}) \right) + r_{jk}(aP) + m_iP$. (Since $\mathscr{C}$ knows the private key x_i corresponding to the nodes N_i, for $i = 1$ to $n-1$.)
  - If $i \neq n$, computes $\sigma_i^{(1)} = \left(\sum_{l \in \mathbb{I}_i} (\sigma_l^{(1)}) \right) + x_i r_{jk}P + m_iP$. (Here, aP is taken from the CDH problem.)
- Computes $F_i = f_iP$, stores the tuple $\langle \sigma_i^{(1)}, f_i, F_i \rangle$ in the list L_F and computes $\sigma_i^{(2)} = y_i f_i P$. (Since $\mathscr{C}$ knows the private key y_i corresponding to all the nodes N_i, for $i = 1$ to n.)
- The signature $\sigma_i = \langle \sigma_i^{(1)}, \sigma_i^{(2)} \rangle$ is returned to $\mathscr{F}$ a the aggregate signature by node N_i.

$\mathscr{O}_{MessageRecover}(\sigma_{Agg}, S, w_j, k)$**:** In order to respond to this query $\mathscr{C}$ performs the following:

- $\mathscr{C}$ checks for a tuple of the form $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H and retrieves the corresponding r_{jk}.
- $\mathscr{C}$ computes $X = \sum_{i=1}^{n-1} x_i(r_{jk}P) + r_{jk}aP$.
- Computes $\sum m_iP = \sigma_{Agg} - X$.
- Solves DL on $\sum m_iP$ with respect to P, retrieves $\sum m_i$ and returns it to $\mathscr{F}$.

Forgery: At the end of the ***Training Phase***, $\mathscr{F}$ produces a valid signature $\sigma_n^* = \langle \sigma_n^{*(1)}, \sigma_n^{*(2)} \rangle$ for the session identified by w_j and the signature number in the session w_j as k. $\mathscr{C}$ computes the solution for the CDH problem from σ_n^* as follows:

- $\mathscr{C}$ checks for the tuple of the form $\langle w_j, k, r_{jk}, H_{jk} \rangle$ in the list L_H and retrieves r_{jk}. ($\mathscr{C}$ knows that $\sigma_n^{*(1)} = \sum_{i=1}^{n} x_i(r_{jk}bP)$.)
- Since the private key of the server S was set to be $\sum_{i=1}^{n-1} x_i + aP$ and $\mathscr{O}_H$ corresponding to (w_j, k) was set to be $r_{jk}bP$, $\mathscr{C}$ computes $X = abP + \sum_{i=1}^{n} m_iP = \sigma_n^{*(1)} - \sum_{i=1}^{n-1} x_i(r_{jk}bP)$.

- $\mathscr{C}$ computes $\hat{e}(P,P)^{\sum_{i=1}^{n} m_i} = \hat{e}(\sigma_{Agg}, P)\hat{e}(\sum_{i=1}^{n-1} x_iP + aP, r_{jk}bP)^{-1}$.
- Solves DL on $\hat{e}(P,P)^{\sum_{i=1}^{n} m_i}$ with respect to $\hat{e}(P,P)$, to find $\sum_{i=1}^{n} m_i$.
- Computes $abP = r_{jk}^{-1}(\sigma_{Agg} - (\sum_{i=1}^{n} m_i)P - \sum_{i=1}^{n-1} x_i(r_{jk}bP))$ and outputs it as the solution for the CDH problem instance.

Thus, $\mathscr{C}$ solves the CDH problem with non-negligible probability.

5.3 Confidentiality

Theorem 5.3. *The aggregate signature with message aggregation scheme AG_MA offers perfect secrecy.*

The condition for perfect secrecy is $Pr[m] = Pr[m|\sigma_i^{(1)}]$ for all $m \in \mathbb{M}$ and $\sigma_i^{(1)} \in \mathbb{C}$, where $\mathbb{M}$ and $\mathbb{C}$ are the message space and the ciphertext space respectively. Since the ciphertext component responsible for the confidentiality of the scheme is $\sigma_i^{(1)}$ and we know that $\sigma_i^{(1)} \in \mathbb{G}_1$.

- The probability that a random $\sigma_i^{(1)}$ will be in the range of encryption is $\frac{|\mathbb{M}|}{|\mathbb{G}_1|}$.
- Probability that $\sigma_i^{(1)}$ is the image of a specific message $m \in \mathbb{M}$ is $\frac{1}{|\mathbb{M}|}$.
- Therefore, $Pr[\sigma_i^{(1)}|m] = \left[\frac{|\mathbb{M}|}{|\mathbb{G}_1|}\right]\left[\frac{1}{|\mathbb{M}|}\right] = \left[\frac{1}{|\mathbb{G}_1|}\right]$.
- Since $\sigma_i^{(1)} \in \mathbb{G}_1$, $Pr[\sigma_i^{(1)}] = \frac{1}{|\mathbb{G}_1|}$.

The condition for perfect secrecy is $Pr[m] = Pr[m|\sigma_i^{(1)}]$, applying **Baye's Theorem** (If $Pr[y] > 0$, then $Pr[x|y] = \frac{Pr[x]Pr[y|x]}{Pr[y]}$.), we get the following:

$$Pr[m|\sigma_i^{(1)}] = \frac{Pr[m]Pr[\sigma_i^{(1)}|m]}{Pr[\sigma_i^{(1)}]} = \frac{Pr[m]\left[\frac{1}{|\mathbb{G}_1|}\right]}{\left[\frac{1}{|\mathbb{G}_1|}\right]} = Pr[m]$$

$\Longrightarrow$ Perfect Secrecy. □

6 Conclusion

In this paper, we have studied the problem of traffic aggregation in a network. We have formalize this problem and suggested a solution using a variant of aggregate signature scheme. We have proved the security of the scheme using standard hardness assumptions in the random oracle model. In the current model, we assume that each node is honest but curious and hence consider that all the nodes participate in the protocol with honest individual traffic values. Thus, our model captures the adversary who is an outsider and tries to know the total traffic or individual traffic of a node and also captures the adversary who is an insider and tries to know the individual traffic of other nodes.

References

[1] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. of the 23th Annual Symposium on Foundations of Computer Science (SFCS'08), Chicago, Illinois, USA*. IEEE, November 1982, pp. 160–164.

[2] "Id-based aggregate signatures from bilinear pairings," in *Proc. of the 4th International Conference, Cryptology and Network Security (CANS'05), Xiamen, China, LNCS*, vol. 3810. Springer-Verlag, December 2005, pp. 110–119.

[3] J. Y. Hwang, D. H. Lee, and M. Yung, "Universal forgery of the identity-based sequential aggregate signature scheme," in *Proc. of the ACM Symposium on Information, Computer and Communications Security (ASIACCS'09), Sydney, Australia*. ACM, March 2009, pp. 157–160.

[4] X. Cheng, J. Liu, and X. Wang, "Identity-based aggregate and verifiably encrypted signatures from bilinear pairing," in *Proc. of the International Conference Computational Science and Its Applications,(ICCSA'05), Singapore, LNCS*, vol. 3483. Springer-Verlag, May 2005, pp. 1046–1054.

[5] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," in *Proc. of the 9th International Conference on Theory and Practice of Public-Key Cryptography (PKC'06), New York, USA, LNCS*, vol. 3958. Springer-Verlag, April 2006, pp. 257–0273.

[6] G. Neven, "Efficient sequential aggregate signed data," in *Proc. of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'08), Istanbul, Turkey, LNCS*, vol. 4965. Springer-Verlag, April 2008, pp. 52–69.

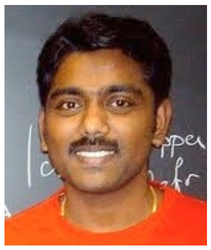
[7] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," in *Proc. of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'04), Interlaken, Switzerland, LNCS*, vol. 3027. Springer-Verlag, May 2004, pp. 74–90.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. of The International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03), Warsaw, Poland, LNCS*, vol. 2656. Springer-Verlag, May 2003, pp. 416–432.

[9] Y. Wen and J. Ma, "An aggregate signature scheme with constant pairing operations," in *Proc. of the International Conference on Computer Science and Software Engineering (CSSE'08), Wuhan, China*. IEEE, December 2008, pp. 830–833.

[10] Z. Wang, H. Chen, D. feng Ye, and Q. Wu, "Practical identity-based aggregate signature scheme from bilinear maps," *Shanghai Jiao Tong University Press*, vol. 13, no. 6, pp. 684–687, December 2008.

[11] J. Herranz, "Deterministic identity-based signatures for partial aggregation," *Computer Journal*, vol. 49, no. 3, pp. 322–330, 2006.



S.Sree Vivek is a PhD scholar in Indian Institute of Technology - Madras, Chennai, India. He is working under the guidance of Prof C.Pandu Rangan. His areas of interest are design and analysis of Identity Based Cryptosystem, Cryptanalysis of cryptosystem, Key Agreement and works on signcryption schemes. His thesis research topic is studies on some encryption, signature and signcryption schemes.

S.Sharmila Deva Selvi is a PhD scholar in Indian Institute of Technology - Madras, Chennai, India. She is working under the guidance of Prof C.Pandu Rangan. Her areas of interest are Provable Security of Public Key Cryptosystem, Cryptanalysis of Identity Based and Certificateless cryptosystem and her works are mostly on the provable security of various flavors of signcryption schemes.

Ramarathnam Venkatesan is a Principal Researcher in Microsoft Research located in Redmond. His interests are mathematical and practical aspects of Complexity Theory, Cryptography and Cryptanalysis, Algorithms and Security. Recently he has been working on Machine Learning, Data Mining, Program Analysis and Synthesis from an adversarial/security view point.

C.Pandu Rangan is a Professor in the department of computer science and engineering of Indian Institute of Technology - Madras, Chennai, India. He heads the Theoretical Computer Science Lab in IIT Madras. His areas of interest are in theoretical computer science mainly focusing on Cryptography, Algorithms and Data Structures, Game Theory, Graph Theory and Distributed Computing.