Secure Data Aggregation using Commitment Schemes and Quasi Commutative Functions

Manik Raina, Subhas Ghosh, Ranjeet Patro, G. Viswanath, Chadrashekhar T
Honeywell Technology Solutions Laboratory
151/1, Doraisanipalya, Bannerghatta Road,
Bangalore, India, 560076
Email:manik.raina@honeywell.com subhas.kumar@honeywell.com

***Abstract*— A scheme is proposed for secure data aggregation in wireless sensor networks. Commitment schemes and a class of functions called *quasi commutative functions* are used to achieve provably secure data aggregation. Efficient schemes to verify the data aggregation are provided.**

I. KEYWORDS

Wireless Sensor Networks, Secure Data Aggregation, Commitment Schemes, Quasi-commutative Functions.

II. INTRODUCTION

Consider a *mesh* wireless sensor network in which there are several nodes. This network is event driven, where potentially each node in the network collects data measurement of the enviromental phenomenon. Certain nodes, in addition to collecting data act as *aggregators*. The data aggregation can be visualized to be happening over an *aggregation tree*. Different events could lead to different aggregation trees. It is assumed that the aggregation tree is already formed (and known to the nodes in the network) i.e our task is to securely aggregate the data, given an aggregation tree. It is assumed that the aggregation begins by a query spreading in the network, followed by the actual aggregation. A scheme is proposed for secure data aggregation in which each party (node of the wireless sensor network) commits to data and some properties the query function (for example the checksum) and then the query function is evaluated. It is assumed that a function $f(.)$ is to be computed on the data $\{d_1, d_2, d_3, ...d_n\}$, the data input of parties $\{p_1, p_2, ...p_n\}$ respectively. It is assumed that this function $f(d_1, d_2, d_3, ...d_n)$ shows the following property

$$f(d_1, d_2, d_3, ...d_n) = f(d_1, f(d_2, d_3, d_4), d_5, ..., d_n)$$

Intuitively, that means that the function of n arguments can be computed by calculating it's value over a few inputs at a time and then putting the results together which is how many aggregation functions (like MIN, MAX, MEDIAN etc) work. In addition to tampering with the results of the data aggregation, we consider another attack where some intermediate dishonest party tampers with the query when it is percolating the sensor network. This is one of the novelties of this paper as many related works assume that the query itself reaches all nodes without being modified. We propose an efficient scheme where parties make non-repudiable *commitments* to their data values and some properties of the query function. By making commitments to the data, parties which cheat on the results of the aggregation can be efficiently detected. By making commitments on the query properties, parties cannot later blame a *tampered* query for incorrect computation of aggregation results by that party. The commitment scheme used in this paper can be used later to verify correctness of computation performed by the parties efficiently. The correctness of the computation at each party can be verified by checking it's data inputs from it's children in the aggregation tree and recomputing the partial function and checking the resultant value against the commitments made. For checking a few nodes against their commitment, this scheme requires little verification overhead. The overhead for a full verification over the whole aggregation tree is discussed in future sections.

This paper is organized as follows. Section(III) introduces notations and cryptographic primitives used throughout this paper. Section(IV) discusses previous works in this are of research. Section(V) defines the problem. Section(VII) discusses the initial setup required while Section(VI) discusses the security goals and adversary's attack model. Section(VIII) discusses the protocol proposed in this paper. The security analysis of the proposed protocol is shown in section(IX). The paper is concluded in section(X) and some limitations of the proposed approach are discussed.

III. Preliminaries

DEFINITION 1 *A function* $h : X \times Y \rightarrow X$ *is* quasi-commutative *[BdM94] if* $\forall x \in X$ *and* $y_1, y_2 \in Y$ $h(h(x, y_1), y_2) = h(h(x, y_2), y_1)$

DEFINITION 2 *A commitment scheme [Blu83][Gol04] is a two-phase two-party protocol in which the party designated as a* sender *can commit itself during the COMMIT phase to a value* $\sigma \in \{0, 1\}$ *so that the following requirements are met*

- Secrecy*: At the end of the COMMIT phase, the party designated as the* receiver *does not gain any information of the sender's value* σ.
- Unambiguity*: It is not possible for the sender to repudiate his choice* σ *during the REVEAL phase*

For a choice $\sigma \in \{0, 1\}$, the sender sends a string $x \in C(\sigma)$ to the receiver such that $C(0) \cap C(1) = \{\Phi\}$ and strings in $C(0)$ and $C(1)$ are computationally indistinguishable for the receiver.

IV. Previous works

Perrig [PSP03] propose a probabilistic scheme for secure data aggregation based on random sampling where a subset of the aggregated data returns to the base station to be verified. This verification is correct within certain bounds of probability. They construct methods for random sampling and interactive proofs for enabling a user to verify a query's accuracy. In their scheme, a query can be verified to a satisfactorily close approximation of the true value even in the presence of cheating a cheating aggregator and some corrupt nodes.

On the other hand, only a constant length commitment is needed to reach the party which originates the query (rather than any data) in our paper. Further, our approach considers a previously unknown attack, where the intermediate adversary corrupts the query as it travels along the network. Verification in our scheme involves verifying computation at a particular node and the commitments of the data of it's immediate children in the aggregation tree.

V. Problem statement

Given a mesh sensor network, the *base station* or another sensor in the network may issue a query to securely aggregate data. The query travels along a *aggregation tree* whose construction and properties are beyond the scope of this work. Some malicious nodes may tamper the query as it travels through the network. All the *leaf* nodes in the aggregation tree pass data to their parents and the *non-leaf* nodes act as aggregators of the data in addition to producing data of their own. The aggregators compute the function $f(.)$ on the received data. The resultant value becomes the value of the non-leaf node and the process repeats up the aggregation tree. The function $f(.)$ must be securely computed with respect to the attacks mentioned in the next section. Further, after the aggregation is performed and the result sent to the originator of the query, the originator must be able to verify the results of aggregation at a party in a small number of steps.

VI. Attack model and security goals

We concentrate on attacks where an adversary who has taken over a node in the network arbitrarily deviates from the computation of the aggregation function by changing the values of the function computation or by tampering with the query which is propogated. In both cases, our protocol ensures *verifiability* and *non-repudiation*, i.e though a party may cheat, it's dishonesty can be detected by a subsequent efficient verification.

VII. Initial setup

We assume that each party shares a secret cryptographic key with it's parent and children in the aggregation tree. Thus, all communication is cryptographically secure. Further, we assume that each party is aware of a rigid integer N and integer X which are used for operations by the quasi-commutative functions as described later. Each party must have a pair of random primes s and t for constructing a number $n = st$ for computing the Rabin function.

VIII. Proposed protocol

A. Elements of the commitment scheme

Our commitment scheme uses the modular exponentiation function([BdM94]) $e_n(x, y) = x^y \ mod \ n$ which is assumed to be a candidate one-way function and considered non-invertible in polynomial time except in an exponentially vanishing number of cases.

A *safe prime* p can be written as $p = 2p^{'} + 1$ where $p^{'}$ is an odd prime. An integer n is called *rigid* if it can be written as $n = pq$ where p and q are safe primes and $|p| = |q|$. Root finding is hard for such an integer n and we assume that every node has such an integer present during initial setup.

Suppose $\{p_1, p_2, p_3, ...p_k\}$ are the k parties who share a common parent in the aggregation tree. If the respective parties want to commit data values $\{d_1, d_2, ...d_k\}$, then each party sends the above values to the parent. The parent calculates $v_p = X^{d_p \prod_{1 \leq i \leq k} d_i} mod \ n$, where d_p is the data of the parent node. X is a predetermined value known to all parties before the protocol begins and n is a *rigid integer*. v_p above could be evaluated as

$$(((X^{d_i} \ mod \ n)^{d_2} \ mod \ n) \ldots^{d_k} mod \ n)^{d_p} mod \ n$$

Since this function preserves the length, the root of the aggregation tree is left with an integer $Y \in Z_n$. Further, each party p_j maintains a partial hash $z_j = X^{Y/d_j} mod \ n$. Verification at a particular party requires verifying that the following property holds $Y = z_j^{d_j} mod \ n$.

B. Protocol

Our protocol is divided into three phases after the query has trickled down the aggregation tree, namely the commit, reveal and verify phases.

During the commit phase, each party commits it's data value and hence cannot repudiate later. Further, each party also commits to some property of the received query U_i (for example, the checksum of the specification of the function $f(.)$ to be computed on the data). The data of each party is not revealed, as the commitment is made using a hard to invert (but polynomially computable) function like quadratic residue. Hence, if party p_i wants to commit data d_i, the commitment is made on $C_i = d_i^2 \ mod \ N_i$, where $N_i = s_i t_i$ where s_i and t_i are primes chosen by each party. We run the algorithm shown below at every party during this phase.

Commitment of data and query properties during COMMIT phase
let v be a vertex in the aggregation tree T on which the protocol runs
if v has no children in T **then**
  Send the commitment $G_v = C_v|U_v$ to the parent of v in T
else
  Gather commitments $G_1, G_2, .., G_k$ from the children of v in T
  Compute $R = X^{G_p \prod_{1 \leq i \leq k} G_i} mod N$
  Send R to the parent of v in T
end if

By the end of the protocol, the root of the aggregation tree will have $Y \in Z_N$. Each party needs to compute $z_i = X^{Y/G_i} \ mod \ N$.

During the reveal phase, each party p_i reveals the data d_i, N_i and U_i to it's parent as shown in algorithm shown below. Further, each party begins computing the function $f()$ on the data and transfers the output to it's parent. As this happens, another commitment process begins with each party committing to it's parent the output of the function at that party. Another commitment tree is built up, this time over the values of the function, as shown below.

Computation of function $f(.)$ and it's commitment along with revealing parameters during the REVEAL phase
let v be the a vertex in the aggregation tree T on which the protocol runs
if v has no children in T **then**
  Send d_v, N_v and U_v to the parent of v in T
else
  Send d_v, N_v and U_v to the parent of v in T
  Compute $O_v = f(d_1, d_2, ..., d_k)$, where d_i is the data from the i'th child of v in T
  Send O_v to the parent of v in T
  if Children of v in T are non-leaf nodes **then**
    Gather $O_1, O_2, ..., O_k$ from the children
    Compute $R^{'} = X^{O_v \prod_{1 \leq i \leq k} O_i} modN$
    Send $R^{'}$ to the parent of v in T
  end if
end if

By the end of this algorithm, the root of the aggregation tree has computed the result of the function and it has two commitment trees, one for the data at each party at the beginning of the protocol (Y) and the tree for the intermediate results of the function computation ($Y^{'}$). Each party computes $z_i^{'} = X^{Y^{'}/O_i} \ mod \ N$ or $z_i^{'} = X^{Y^{'}/d_i} \ mod \ N$

depending on whether it is a leaf node or not.

During the verify phase, the check on the correctness of computation is performed. To verify the correctness of any intermediate computation at any party p_l, the inputs provided by the children of p_l must be ascertained for correctness and so should be the inputs of p_l itself. The first can be achieved by querying the every child of p_l in the aggregation tree T, to determine it's committed data (if it is a leaf node) or the output of the function at that party (if it's a non leaf node) is valid as per commitment. The following step would have to be executed for every child of p_l in T as shown in algorithm as shown below.

Verifying commitments of children of party p_l
for all $p_r \in$ children(p_l) **do**
  if p_r is leaf **then**
    $C_r = d_r^2 \ mod \ N_r$
    check if $z_r^{C_r|U_r} \ mod \ N = Y$
  else
    check if $(z_r^{'})^{O_r} \ mod \ N = Y^{'}$
  end if
end for

After this step, the computation at p_l itself is checked, which includes checking the local input and the computation itself as shown in algorithm below.

Verifying commitments of party p_l **and it's computation**
$C_l = d_l^2 \ mod \ N_l$
if p_l is leaf node in T **then**
  Check if $z_l^{C_l|U_l} \ mod \ N = Y$
else
  Check if $z_l^{O_l|U_l} \ mod \ N = Y$
  for all $p_r \in$ children(p_l) **do**
    if p_r is leaf node in T **then**
      $val_r = d_r$
    else
      $val_r = O_r$
    end if
  end for
  check if $O_l = f(val_1, \ldots, val_{|children(p_l)|})$
  check if $(z_l^{'})^{O_l} \ mod \ N = Y^{'}$
end if

The overhead of verification of this protocol depends on the topology of the Wireless sensor network. To verify the complete aggregation tree, we would need queries between $log\ m$ and m where m is the number of sensors in the network.

IX. Security analysis

The non-invertibility of the quasi-commutative function used is assumed. On repeated application of the modular exponentiation function N must be chosen such that the possibility of finding collisions is reduced. As described in section 8.1, N must be a *rigid integer* following the properties that $N = pq$, where p and q are two primes such that $p = 2q + 1$ and $|p| = |q|$. If the factorization of N is hidden, then it's extremely improbable that repeated application of the same function will result in a reduction in the size of the domain or produce random collisions. The computational hardness we want to achieve is - given a partial accumulated hash z_i and data y_i such that $z_i^{y_i} \ mod \ N = z$, it is computationally infeasible for an adversary to determine a $z_i^{'}$ in polynomial time such that for a forged $y_i^{'}$ the following holds true

$$(z_i^{'})^{y_i^{'}} \ mod \ N = z$$

Where all z_i, y_i etc $\in Z_N$. Shamir [Sha81] shows that for an appropriately chosen N, if finding the root modulo N is hard, then the modular exponentiation function forms a family of one-way functions. As pointed out in [BdM94] it is noteworthy that even though construction of rigid integers is harder than constructing hard to factor integers, it is still quite feasible. More importantly, we need to generate a rigid integer N which can be done offline and conveyed to every party securely before the protocol begins.

THEOREM 1
Any party p_i once having committed to data d_i, cannot substitute it with erroneous data $d_i^{'}$ during the computation of the function $f()$ without being detected during the VERIFY phase.
PROOF: As shown in [BdM94], given the modular exponentiation function $e_N(X, y) = X^y mod\ N$ where N is a rigid integer, it is computationally infeasible to determine a number $X^{'}$ given $y^{'} \in Z_N$, such that $e_N(X, y) = e_N(X^{'}, y^{'})$ in polynomial time. This means that, given a partial commitment z and data committed by a party C_i, it is computationally infeasible (in polynomial time) to determine $C_i^{'}$ and $z^{'}$ such that $z^{C_i} \ mod \ N = (z^{'})^{{C_i}^{'}} \ mod \ N$. Hence, once committed to a data value, a cheater cannot repudiate it.

To repudiate it's data commitment, the malicious node would have to invert the Rabin function by determining an integer $C_i^{'}$ such that

$(C_i^{'})^2 mod\ N = C_i^2 mod\ N$, which is at least as hard as inverting a one way function. ■

THEOREM 2

Any party p_i once having committed to a certain query property U_i, cannot repudiate it.

PROOF: Each party commits to some property (U_i) strongly associated with the integrity of the received query, for example the hash of the recived function f to compute over the data. Since a commitment to the query property is made using U_i during the COMMIT phase, to repudiate it would entail determining $U_i^{'}$ and $z_i^{'}$ such that $(z_i^{'})^{C_i|U_i^{'}} mod\ N = z_i^{C_i|U_i} mod\ N$. Doing so is at least as hard as inverting the modular exponentiation function and hence infeasible in polynomial time. ■

THEOREM 3

Any party p_i once having committed to a partial function computation at that party O_i, cannot substitute it with erroneous data $O_i^{'}$ during the computation of the function $f()$ without being detected during the VERIFY phase.

PROOF: As shown in [BdM94], given the modular exponentiation function $e_N(X, y) = X^y\ mod\ N$ where N is a rigid integer, it is computationally infeasible to determine a number $X^{'}$ given $y^{'} \in Z_N$, such that $e_N(X, y) = e_N(X, y^{'})$ in polynomial time. Hence, given a partial commitment z and committed output O_i, it is computationally infeasible in polynomial time to determine $O_i^{'}$ and $z^{'}$ such that $z^{O_i}\ mod\ N = (z^{'})^{O_i^{'}}\ mod\ N$. ■

Let m be the number of sensors in the network. We need m queries to verify the entire aggregation. However, if we suspect a few sensors of cheating, the partial computation at just that sensor could be verified by inspecting the commitments at that sensor and it's children sensors in the aggregation tree.

X. CONCLUSION

We have presented an algorithm for secure data aggregation in wireless sensor networks, discussed it's security properties and proved secrecy of this mechanism. [PSP03] proposes a data aggregation algorithm which gives an ϵ approximation by taking a fraction of the data. Their verification mechanism is based on the PCP based (ϵ, δ) verification mechanism which is probabilistic. Our verification gives certain results (declares a sensor as cheating or not). Further, we can inspect individual (of a fixed set of) sensors which are suspected of cheating. A drawback of our algorithm is that it can be applied to functions which can be partially computed as shown in section(II) and could form the basis for future work.

REFERENCES

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 274–285. Springer-Verlag New York, Inc., 1994.

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[PSP03] Bartosz Przydatek, Dawn Song, and Adrian Perrig. Sia: secure information aggregation in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 255–265. ACM Press, 2003.

[Sha81] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. ICALP, 1981.