The Security - Communication tradeoff in Mobile Computing

Asnat Dadon-Elichai and Danny Raz

Abstract

Mobile computing introduces a paradigm that enables new services to be deployed efficiently over the network. However, these new promising capabilities come with inherent security hazards, both for the machines that host the mobile code, and to the mobile code itself. In this paper we present a framework for the rigorous study of the latter problem, i.e., the ability of the mobile applet to carry the desired computation in a secure and safe manner. Our model considers both the ability of the mobile applet to perform the computation, the amount of information revealed by the environment as a result of this computation, and the communication overhead required.

Using our framework we can show that secure and safe computing can be done. However, this requires a significant communication overhead. Using a very strong cryptographic assumption (like the one proposed by Sanders and Tschudin [ST98]) we present algorithms that reduce the amount of the required communication overhead for some common tasks.

We also present a different scheme, based on fault-tolerant and cryptographic techniques, that does not require the strong requirements. The idea is to verify the correct execution of the computation using distributed “chaperons”. Each of the chaperons verifies part of the computation but none of them have enough information to reveal a significant amount of information regarding the applet’s computation. We prove that our scheme is secure and safe, in the sense that it prevents either party from repudiating its commitments, and detects faulty computation during execution. Another property of our scheme is that it enables the detection of mobile applet code/data tampering and enables one to reveal the identity of possible malicious hosts, if there are any.

Key words: Security, protection, mobile applet, secret-sharing, fault-tolerant, communication.

1 Introduction

Active systems such as the “agent-based systems” and “active networks” are based on the ability to launch an applet to be executed remotely and to act on behalf of the issued service. The mobile applets, mapplets for short, are autonomous entities that migrate within the active system under their own control. ¹

¹This paper refers to mobile code and agents as mapplets.
Active systems are attractive because they present a flexible paradigm that extends traditional system capabilities and improves the overall utilization of system resources. Unfortunately the paradigm also presents new and hard security issues that fall into two main areas: Securing the hosts from malicious mapplets and securing the mapplets from malicious hosts. Both areas are subjected to extensive research.

Security and protection of hosts from malicious mapplets rely on current methodologies such as access control mechanisms and cryptographic techniques. Research in the area of securing mapplets from malicious hosts is considered the most difficult security issue of active systems, because mapplets are under the full control of the foreign host. A malicious host can influence the computation of the mapplet, steal its secrets, take control over its digital money, etc. For example, according to [XWL96], “Current consensus is that it is computationally impossible to protect mobile agents from malicious hosts”.

Current research in the area of securing mapplet execution either does not result in straightforward implementations, or does not prevent malicious hosts from influencing the computation, or limits the active paradigm abilities (e.g. autonomous migration, asynchronous and parallel computation, or reaction to dynamic events, new information and services). Thus, the issue of protecting mapplet computation still remains, and needs to be addressed before the active systems paradigm can be fully deployed.

This paper presents a rigorous framework study of the secure assurance of mapplet computation. The framework accounts for other issues such as the information achieved by the environment as a result of the computation by the mobile applet, and the communication overhead that is required for supporting these capabilities. We first show that even under very mild realistic assumptions, secure remote computation can be carried out. However, the algorithm we present requires the same amount of communication that is required in the traditional client/server model. Such a solution could not practically be accepted as it stands against the main objective of the remote computation paradigm - saving communication along wide area connections. We then show how to use a stronger assumption, like the one presented by Sanders and Tschudin in [ST98], to reduce the amount of required communication while maintaining the computation security. We then introduce a novel computation scheme where the mapplet is escorted by chaperon mapplets that act as a distributed reference point. We use secret sharing techniques from [Sha79] to ensure that the
hosts of these chaperons will not retrieve too much information regarding the mapplet’s computation. Another property of our scheme is that it enables the detection of mobile applet code/data tampering and enables to reveal the identity of possible malicious hosts, if there exists any.

The main contributions of this paper are as follows. This is the first rigorous study that combines the study of the correctness and security of the remote computation with the performance in terms of communication overhead. By indicating the required cryptographic assumption we lay the ground for a formal study of this important field. We also present a novel distributed scheme for remote computation that may allow a low cost secure remote computation in large networks.

The paper is organized as follows: Section 2 details the concepts, assumptions and definitions of the model. Section 3 defines the obvious secured scheme for remote computation and proves its correctness. An improvement over the obvious scheme, one that minimizes the communication overhead is also presented. In section 4 we introduce the distributed secured model and prove its correctness. We also discuss the information achieved by the participant hosts and provide a well specified estimation for assuring the scheme computation by defining the relation between the number of the queries and the number of participant hosts. In section 5 we discuss related works and in section 6 we draw some conclusions and suggest future directions.

2 A Model for Remote Computation Assurance

In this section we describe the assumptions and definitions that are needed for assurance of distributed and secure remote computation. A remote computation is based on launching a mapplet to be executed on some remote host on behalf of the initiating service. A secure computation in this context is one that assures that any commitment provided by all parties will not be repudiated at a future time, and that the integrity of the computation is supported. Consider the following basic scenario described in Figure 1. An originator $O$, needs to perform a computation based on some data from a different host $H$. This can be done either in the traditional client/server model which we generally refer to as distributed computation, or by a mapplet $m$, that is being executed at the host $H$, which is referred to as remote computation.

We propose an approach that provides a formal overhead analysis of realistic schemes. In order to make it meaningful we begin with a formal description of our assumptions regarding the various
elements in the systems.

2.1 Assumptions

The first eight assumptions are generally needed to analyze the security of a distributed and/or remote computation. The last assumption is a much stronger one about the ability of executing a secure computation at a possibly malicious host. This assumption was considered to be essential for such a computation as described in [ST98].

Assumption 2.1. Upon relocation, the mapplet m, can establish a reliable communication with O by using a suitable communication protocol.

This assumption describes the communication mechanism used by Java applets. When a Java applet is downloaded to the end system it may open a TCP/IP connection to its source.

Assumption 2.2. O is trustworthy to m, it “knows” the correct algorithm of m.

This assumption reflects the fact that O, the originator of m, is the most reliable reference point for m.

Assumption 2.3. Every message that is sent to H by O or m, is signed.

This assumption comes to eliminate cases where O will repudiate a commitment it gave to H directly or indirectly during the execution of a remote computation.

Assumption 2.4. There exists a well defined API, for a communication between mapplets and the local hosts.
**Assumption 2.5.** \( H \) signs every data it produces by using its private key.

This assumption prevents \( H \) from repudiating any commitment that it had made.

**Assumption 2.6.** Every host, including \( O \) and \( H \), can authenticate any message they receive.

Each entity digitally signs all its messages by using its private key. \( H \) signs every message based on assumption 2.5 and \( O \) can safely sign its messages based on assumption 2.3.

**Assumption 2.7.** \( H \)'s only knowledge is that \( m \) has decided to migrate to it. \( H \) has no former knowledge on \( m \)'s history computation results.

This assumption allows to ignore cases where \( H \) has previous knowledge on \( m \)'s task. Previous knowledge enables \( H \) to present faulty results in advance.

**Assumption 2.8.** All the hosts included in the active system are polynomial time bounded.

This assumption assures that cryptographic techniques can be used without considering cases where hosts can forge another host’s signature.

**Assumption 2.9.** [Also referred as the Black-box assumption]: The execution of a given mapplet, \( m \), is considered a black box for the foreign host \( H \). \( H \) cannot “understand” nor can it interfere with \( m \)'s computations.

In 1998, Sanders and Tschudin’s argued that protecting mapplet computation can be provided by using cryptographic methods [ST98]. They presented a technique called Computation with Encrypted Function, (CEF), where the mapplet code is encrypted before being launched to be executed at the host. Due to the fact that decomposing an encrypted function is considered a hard problem, the host \( H \) executes the mapplet as a black box. Thus it cannot understand nor influence \( m \)'s activity.

Assuming this assumption, the following two observations hold:

**Observation 2.10.** \( H \) cannot decrypt nor intercept encrypted communication between \( m \) and \( O \).

Since the mapplet, \( m \) can be executed as a black-box, it can encrypt the communication. Since \( H \) cannot access \( m \)'s private key it cannot intercept nor decrypt the communication.
Observation 2.11. The mapplet $m$ has the ability to authenticate whether a given message was initiated from either $O$ or $H$.

Signatures are provided by using private keys on a digested message. Signature authentication and validation is provided by using the public key and the signed text. Since public keys are publicly known and since $m$’s execution cannot be influenced by $H$, Observation 2.11 follows.

In this paper we examine the actual need for the black-box Assumption. It is currently widely accepted that this assumption is essential to assure the secured computation of a mapplet. Later on we show how one can actually maintain the computation integrity without relying on this assumption.

2.2 Definitions

The following definitions are used in order to properly formulate the security and integrity of a mapplet computation.

Definition 2.12. **External effect** of a mapplet $m$ is a commitment that $m$ gave to $H$ or any change applied on the database of $O$.

*External-effects* are commitments that a mapplet $m$ delivered to $H$ during the remote computation and that $O$ is obligated to honor.

Definition 2.13. A computation of $m$ is **correct** if upon termination, its external effects are identical to those that would have been received if $m$ would be executed by a trustworthy host.

The correctness of a remote computation is measured by the commitment set that was delivered to $H$ while the computation took place. If the exact set of commitments is gained both in a local and a remote computation, then the environment did not play any role on the computation.

Definition 2.14. A given commitment $C$ is **satisfactory** with respect to an external effect $ea$ if there exists a computation of $m$ in $O$ that results in the same $ea$ in which $H$ committed to $C$, where $O$ is $m$’s originator.

By this definition we set the relationship between the commitments of $H$ to $O$ and the external effects that were delivered to $H$ or stated differently, different commitments are produced by $O$ as a result
of different commitments of $H$. In addition there is no 1:1 relation between the commitments of $O$ and $H$. For example, $H$ may commit to several proposals with different terms and $O$ can commit to purchase only one proposal.

**Definition 2.15.** A computation of $m$ is **safe**, if upon termination, $H$’s commitments to $O$ are satisfactory.

This definition defines the relation between the commitments of $O$ to the commitments produced by $H$.

**Definition 2.16.** $m$ is **not-revealing** if for every execution of $m$ in $H$ in which as a result $H$ acquired information $K$ and provided a satisfactory commitment $C$, there exists a distributed computation in which $H$ acquires $K$ and provides $C$.

One of the hazards of remote computation is that a host can “understand” the mapplet computation goals and influence the computation to favor its own goals. We argue that if the same information is concluded by the foreign host with regardless if the computation was carried either in a distributed or remote fashion then both models are equally revealing and $H$ will influence the computation equally.

**Definition 2.17.** A mapplet $m$ is **good** if it is correct, safe and not-revealing.

**Definition 2.18.** $Comm(O, m, H)$ is the number of bits required for a correct, safe and not-revealing computation of $m$.

### 3 A Simple Scheme for a Secure Remote Computation

In this section we study the basic problem regarding the issue of a secure remote computation. Our first observation is that every distributed computation can be done in a secure and safe way by a mapplet via remote computing. This can be done based on the basic assumptions without the need for the black-box Assumption. Once this is established, we turn to study the amount of communication really needed in order to have a secure and safe remote computation. For this we use the additional strong black-box Assumption. We end this section with a couple of non-trivial examples that show how one can save bandwidth while maintaining a good computation.
3.1 The existence of a good remote computation

We now turn to answer, affirmatively, the basic question of the existence of a good remote computation.

Consider the mapplet (called Obvious or Obvs) that works as follows. The originator $O$ sends a transaction to Obvs instead of sending it directly to $H$. Obvs submits the transaction to $H$ via $H$'s local supported interface. $H$ computes its response and sends the signed result to Obvs, which immediately forwards it to $O$. Obvs terminates its execution when it receives an explicit instruction from $O$.

Figure 1 presents the suggested configuration. Figure 2 formally describes the algorithm.

```plaintext
mapplet Obvs()
    Home ← OpenConct2Host(HomeAddress);
    SendMsg(Home, getHostAddress());
    while (TRUE)
        M ← ReceiveMsg(Home)
        where M = (Action, Data);
        if (Action = Terminate)
            TakeTerminateActions();
            selfTerminate();
            SubmitMsg2HostViaAPI(M);
            M ← ReceiveResultsfromH();
            SendMsg2Host(Home, M);
    mappletEnd Obvs;
```

Figure 2: Obvs - The Obvious Algorithm

**Theorem 3.1.** Obvs is **good**.

**Proof.** Based on definition 2.17, a mapplet is **good** if it is **correct**, **safe** and **not-revealing**. Since Obvs imitates the distributed computing, it follows immediately from the above definitions that it is **correct**, **safe**, and **not-revealing**. □

As mentioned above, the **black-box** Assumption is not needed in this case, since both Obvs algorithm and all the messages exchanged between Obvs and $O$ are eventually known to $H$. Since Obvs is universal we can even further assume that it is provided by the target host upon computation initiation. Thus, to pinpoint the argument the following observation is provided.
Observation 3.2. A good remote computation exists even without the black-box Assumption.

Note that our basic observation only deals with the existence of a good mapplet according to our definitions. Our scheme does not address the ”normal” security hazards that might be affecting the host $H$ as a result of a malicious mapplet. The next step is to analyze the amount of communication that is inherently needed to carry out a good computation. Mapplets like Obvs use at least the same amount of communication as the distributed computation, and the overall goal of using remote computation is to reduce the communication needed. In the next section we formally describe the communication overhead of a computation and we study different ways to reduce this amount while maintaining the computation’s “goodness”.

3.2 An improvement of Obvs algorithm

In order to formally study the security-communication tradeoff we need to formally define the communication overhead of a mapplet.

**Definition 3.3.** Given an $O$, $m$ and $H$, the overall communication, $\text{Comm}(O, m, H)$, is defined as the amount of communication (in bits times hops) needed to accomplish a given task, by using remote computation of $m$ in $H$.

**Definition 3.4.** A Communication ratio of $m$, $C_r(m)$, is $r$ if there exists a constant $c$ such that for every $H$ and $O$ the following holds: $\text{Comm}(O, m, H) \leq \text{Comm}(O, Obvs, H) \cdot r + c$ where Obvs is the obvious good $m$.

The question that we are studying can be stated now as follows: Given the above definitions and the model, is there a solution that solves the problem with less overhead while not compromising the computation integrity? I.e., can we generate a good remote algorithm $X$ such that $C_r(X) << C_r(Obvs)$?

To answer this question we need to consider both the communication between $O$, $m$ and $H$, and the goodness of the remote computation.

For the first issue we can either reduce the distance that the exchanged messages have to travel or reduce the overall amount of communication. The above implies that we now consider that some of the transactions and/or results will reside in the mapplet’s data segment and that the mapplet
will carry a partial computation autonomously. Due to this, we first assume that the *black-box* Assumption holds.

Assuming that the computation cost of *Obvs* in *H* is negligible it is clear that the main cost of the scheme presented in section 3 results from the communication between *O* and *Obvs*. From a security point of view, the transactions that need special protection are commitments that will be submitted to *H* during the computation and that *O* is obligated to honor, since commitments are the main target for tampering by the foreign hosts. Thus, it is wise that these commitments be issued by *O* itself and not by a mapplet. Another argument to favor this approach is motivated when considering security breaches such as blocking messages. This kind of hazard prevents *H*’s commitments from reaching *O* while enabling it to get hold on the *external effects*.

Thus, the transactions delivered to *H* are classified into two logical kinds of transactions: information retrieval and *external effects*. The following notation is used to distinct the two kinds of transactions:

- $ir_x$: information retrieval request for $x = 1 \ldots k$.
- $cd_x$: decided commitment for $x = 1 \ldots k$.

Using this classification the overall messages that stream from *O* to *H* during the remote computation\(^2\) are divided into a series of blocks where every block contains either one decided commitment or a set of information retrieval transactions. i.e. $B_i = \{cd_x | ir_{i_1} \ldots ir_{i_l}\}$ \forall i.

Thus, by assuming that $n$ transactions are transmitted from *O* to *H* for solving a given problem, $TS(O, m, H) = \{t_1 \ldots t_n\}$, then this set of transactions is viewed as follows: $TS(O, m, H) = \{B_1 \ldots B_r\}$ where $i = 1 \ldots r$ denotes the number of the transmitted blocks from *O* to *m* in *H*. It is also clear that:

$$\sum_{i=1}^r |B_i| = n$$

(1)

where $|B|$ defines the number of transactions included within block $B$.

Classifying the transactions delivered from *O* to *H* as described above, enables one to design an improved algorithm for remote computation while maintaining the interactive behavior of a

---

\(^2\)We would like to emphasize that the message stream between *O* and *m* may include only a subset of *O*’s transaction domain. Like any application the transactions produced by *O* are the result of its internal status, and the received input.
mapplet causing the reduction of the communication overhead. The ImpObvs algorithm presented in figure 4 is an improved version of the Obvs algorithm. The ImpObvs algorithm uses the above detailed considerations to reduce the communication overhead while maintaining the goodness of the remote computation.

3.2.1 ImpObvs algorithm

Taking the above into account, the improved algorithm, ImpObvs, is designed to carry its computation by using two kinds of behavior, as a result of the message it receives from $O$. When ImpObvs receives a commitment block from $O$ it immediately submits the included transaction to $H$ and redirects $H'$'s response to $O$.

This kind of behavior resembles Obvs's behavior and is referred as interactive. If on the other hand it receives an information retrieval block, it submits all and/or subsets of the included transactions to $H$, according to some predefined algorithm, performs some computation on $H'$'s results and finally transmits itself signed computation to $O$. We refer to this kind of behavior as an autonomous one.

Thus, after ImpObvs informs $O$ that it has been successfully relocated on a foreign host, it continues as described above, and like Obvs it terminates upon receiving an explicit instruction from $O$. In case an execution error is encountered during the computation ImpObvs immediately informs $O$ and continues its execution based on $O$'s explicit instructions. I.e. terminates or migrates to another host. The suggested algorithm is extended to include some checking points that are on used by a mapplet to inform its originator with intermediate results of the autonomous computation. This optional extension is emphasized within the code. In figure 4 the ImpObvs algorithm is presented.

It is worth to note that the actual location of ImpObvs has no importance from the goodness of computation point of view. ImpObvs can be relocated on any host and starts its execution. Figure 3 describes the possible configurations. Let $H'$ be the host where ImpObvs is relocated and executed. Let $H$ be the host where the data that ImpObvs has to process is located. In case where $H' \neq H$ then upon ImpObvs relocation on $H'$ it opens a connection to $H$ and submits the transactions via the opened connection, in the same manner as of the distribute computation model. In the case where $H' = H$ then the opened session is local. Figure 3 describes the possible configuration for
remote computation.

![Diagram of possible configurations for ImpObvs location](image)

Figure 3: Possible Configurations for ImpObvs location \( H' = \{Origin, Target, otherhost\} \)

From here to the end of this section we prove the goodness of ImpObvs. We first present a definition for interactive computation. Next, we prove that ImpObvs execution results directly from \( O' \)'s instructions and we finalize with proving that ImpObvs is good.

**Definition 3.5. Interactive Execution:** An interactive execution is a series of atomic execution steps that is subjected to the following behavior:

1. **An atomic execution step** that composed of:
   - Local computation of the process.
   - One communication action: either reception or transmission of a message.

2. The communication actions, reception/transmission are interleaved. i.e. any finite series of interleaved inputs and outputs. \([I_0, O_0, I_1, O_1, \ldots]\).

We argue that ImpObvs execution is controlled only by its originator \( O \), and as a result of its explicit instructions. This behaviour of ImpObvs is demonstrated by proving the claims included within the following list. Thus, for every execution of a given \( H' \) and \( O \), where \( H' = H \), the computation of ImpObvs provides the same results.\(^3\) The following are basic observations from the pseudo-code of Figure 4.

- Any message that is received by either \( O \) or ImpObvs, which does not cause ImpObvs to stop its computation is authentic;

---

\(^3\)In case where \( H' \neq H \) then the computation model is equivalent to the distribute computation and thus it is good.
1 mapplet ImpObvs()
2 CurrHostAddr ← getCurrentHostAddress();
3 if not (relocated at required host(CurrHostAddr))
4 Migrate to required host (HostAddress);
5 selfTerminate();
6 Home ← openConnection(HomeAddress);
7 while (TRUE)
8 M ← getMsg(HOME) where M = (Action, Data)
9 if (not validateMsg(Data))
10 SendMsg (Home, (NotValidMsg, M);
11 if (Action = Terminate)
12 SendMsg(Home, (Terminate))
13 SelfTerminate();
14 if (Action = establishConnectionHost )
15 Host ← openConnection(Data);
16 if (Action = Submit2Host )
17 SendMsg (Host, Data);
18 M ← getMsg(Host);
19 SendMsg (Home, (Result, M));
20 if (Action = Autonomous ) where Data = {setLocals(), ir_1 ... ir_n,stopCheckCond}
21 while (there are more queries to process)
22 Transaction ← fetchNextQuery(ir_1 ... ir_n);
23 SendMsg (Host, M) where M ← format(Transaction);
24 M ← getMsg(Host) where M = (_, Hresults);
25 (Action, Data) ← performsComputation(initLocals(), Hresults, stopCheckCond())
26 if (Action = SendToHome)
27 SendMsg (Home, Sign(PrivKey, InterRes(DataSegment)));
28 M ← (Final, Sign(PrivKey, GetImage(DataSegment)));
29 SendMsg (Home, M);
30 mappletEnd ImpObvs;

Figure 4: ImpObvs - An Improved algorithm of secure remote computation
- \textit{ImpObvs} is relocated in the required target host \(H';\)
- \textit{ImpObvs}'s execution is carried only after it opens a secured session to \(O's\) chosen target host \(H;\)
- \textit{ImpObvs}'s default execution is interactive;
- \textit{ImpObvs} changes its mode of execution as a result of an explicit instruction from \(O;\)
- Every autonomous execution of \textit{ImpObvs} is finite and determined by the originator \(O.\)

\textbf{Claim 3.6.} Any message received either by \textit{ImpObvs} or by \(O\) which does not cause an abnormal termination of \textit{ImpObvs} is authentic.

\textit{Proof.} Based on the \textit{black-box} Assumption regarding the execution of \textit{ImpObvs}, \(H'\) cannot intercept nor influence \textit{ImpObvs}'s execution including message signing. Based on Assumption 2.6 \(O\) can validate whether a received message (\(M\)) is authentic.

Based on the Observation 2.10 and Observation 2.11 \textit{ImpObvs} can validate the authentication of \(M\) and this action is not influenced by \(H'.\) Thus if \(O\) or \textit{ImpObvs} cannot authenticate \(M,\) actions are taken to terminate \textit{ImpObvs}'s execution. If on the other hand termination actions are not taken by either peers, then \(M\) is authentic. \hfill \Box

\textbf{Claim 3.7.} \textit{ImpObvs} is relocated to the required target host \(H'.\)

\textit{Proof.} We first assume that the routing system works properly. According to the above algorithm, \textit{ImpObvs} gets the current host address, at line 2, upon relocation. According to Assumption 2.5, \(H'\) signs every response it produces, including its address. \textit{ImpObvs} compares the resulted signed address with the required target host address.

If the current host is not the target host then \textit{ImpObvs} takes actions to migrate to the required host according to line 3 and terminates its execution.

If, on the other hand, the current host is allegedly the required host, \textit{ImpObvs} opens a connection to \(O.\) This action implies that \textit{ImpObvs} was located on the required host \(H'.\) \(O\) receives the address and can validate \(H'\)'s signature. Since only \(H'\) can sign its address properly, \(O\) can decide if \textit{ImpObvs} is actually relocated at the required address. If \(O\) is satisfied with \textit{ImpObvs}'s location, it instructs it to open a secure session to some given host, otherwise it instructs it to terminate. 14
ImpObvs receives the message \((M)\) on line 8 and authenticates it on line 9. Based on Assumption 2.2, \(O\) is trustworthy, executes correctly the algorithm, and sends the proper message. In case \(O\) sends a Terminate command then ImpObvs sends a terminate message to \(O\), according to line 12 and takes actions to terminate itself on line 13, with no interference, based on the black-box Assumption. If on the other hand ImpObvs is not terminated, then it is located on the required target host \(H'\).

**Claim 3.8.** ImpObvs moves to autonomous mode as a result of \(O\)'s explicit instruction.

**Proof.** Let's assume that ImpObvs changed its execution mode without an explicit instruction from \(O\). This means that the change took place during the local execution of ImpObvs. Based on the black-box Assumption \(H'\) cannot alter ImpObvs’s inner state values. Thus the change took place by ImpObvs itself while executing its algorithm. But the only place in the code that causes ImpObvs to execute lines 19-28 results from the value of the variable \(Action\). The value of the variable \(Action\) is extracted at line 7 from an authentic \(M\) that was received from \(O\), in contrast to the initial assumption.

**Claim 3.9.** ImpObvs’s autonomous execution is finite.

**Proof.** Based on Claim 3.8 ImpObvs starts to execute in autonomous mode, as a result of an explicit instruction from \(O\). The message that instructs ImpObvs to change to autonomous mode provides it with an optional setting, a set of information retrieval transactions and an optional condition to stop execution as described on line 20. The optional setting and stop conditions enable the originator \(O\) to control ImpObvs’s execution in a context-free manner\(^4\). ImpObvs fetches one transaction from the provided set on line 21, and sends it to \(H\) in line 22. ImpObvs evaluates the optional conditions provided to it by \(O\) are fulfilled by calling \(performsComputation()\) in line 23. If the conditions are fulfilled then it sends \(O\) its signed data image. The while loop between lines 20-27 is carried until all the transactions included in the set are computed. Since the number of \(O'\)s transactions is finite, it results that the number of the transactions set delivered to ImpObvs is also finite.

**Claim 3.10.** \(O\) is the source of every transaction that ImpObvs submits to \(H\).

\(^4\)A context-free execution enables the originator \(O\) to deliver different transactions with no regard to the mapplet’s real implementation.
Proof. \(\text{Let's assume that a transaction was submitted to } H \text{ without being explicitly initiated by } O.\) Based on Assumption 3.6 and the \textit{black-box} Assumption, every transaction submitted to \(H\) by \(\text{ImpObvs}\) is authentic and results from \(\text{ImpObvs}\)’s correct execution. The two positions in the code that \(\text{ImpObvs}\) submits transactions to \(H\) are located in lines 16 and 22. In line 16 \(\text{ImpObvs}\) submits to \(H\) a transaction that were delivered to it from \(O\) via an authentic message. In line 22 the transaction was selected from a block of transactions that was delivered to it previously by \(O\) also via an authentic message. Thus, there is no place in the algorithm that \(\text{ImpObvs}\) submits to \(H\)’s transactions that were not previously initiated from \(O\).

\begin{claim}
\text{ImpObvs is interactive.}
\end{claim}

Proof. Based on Claim 3.8 \(\text{ImpObvs}\) starts to execute in autonomous mode as a result of an explicit instruction from \(O\). Based on Claim 3.9 the autonomous execution mode is finite and it’s length is defined by \(O\). The autonomous execution of \(\text{ImpObvs}\) is actually subjected to the definition of the atomic computation step that ends with a communication action. Consequently, \(\text{ImpObvs}\) is interactive.

\begin{claim}
\text{ImpObvs is correct.}
\end{claim}

Proof. \(\text{ImpObvs is correct}\) if the same \textit{external effects} are achieved at the end of either the distributed or the remote computation. \(\text{Let's assume that } H \text{ holds an external effect that is not reflected in } O\)’s database. Since the \textit{external effect} is valid only if it is signed by \(O\) and \(H\) does not have neither \(O\)’s nor \(\text{ImpObvs}\)’s private keys and since based on Assumption 2.8 \(H\) cannot forge another ones signature, it results that \(\text{ImpObvs}\) submits the \textit{external effect} to \(H\). Based on the \textit{black-box} Assumption, \(H\) cannot alter \(\text{ImpObvs}\) to submits it with a forged \textit{external effect}. Thus, either \(O\) or \(\text{ImpObvs}\) produced it and submitted it to \(H\). If \(O\) produced the \textit{external effect} than it resulted from a change in its database, since \(O\) is trustworthy. Thus, based on the claim assumption, \(\text{ImpObvs}\) produced the \textit{external effect} autonomously, in contrast to Claim 3.10. Thus the \textit{external effect} was kept in \(\text{ImpObvs}\) and upon relocation it was submitted to \(H\). This case is not possible based on Claim 3.10. Thus the current claim assumption is faulty and \(H\) cannot hold any \textit{external effect} that did not result previously from a change in \(O\)’s database.

\begin{claim}
\text{ImpObvs is safe.}
\end{claim}
Proof. \textit{ImpObvs} is \textit{safe}, if upon termination, any satisfactory commitment \((C)\) \(H\) produces with respect to a given \(ea\) would has been produced either in distributed or remote computation. Lets assume that there exists a computation of \(\text{ImpObvs}\) in a trustworthy host where \(H\) commits to \(C\) and a computation of \(\text{ImpObvs}'\) in \(H\) where \(H\) committed \(C'\) and \(C \neq C'\) with respect to the same \(ea\). Based on Claim 3.10 any transaction including \(ea\), that is submitted to \(H\) by \(\text{ImpObvs}\) was produced by \(O\) to reflect a change in \(O\)'s database. The \(ea\) is produced as a result of \(H\)'s commitments. \(H\) signs any result it produces including commitments. \(H\)'s commitments can be authenticated by using \(H\)'s public key. This means that based on the locations of \(\text{ImpObvs}\), \(O\) produces the same \(ea\) for two different commitments \(C\) and \(C'\). Since \(O\) is trustworthy and follows the algorithm correctly this cannot happen.

\begin{comment}
\begin{proof}
\textit{ImpObvs} is \textit{safe}, if upon termination, any satisfactory commitment \((C)\) \(H\) produces with respect to a given \(ea\) would have been produced either in distributed or remote computation. Let's assume that there exists a computation of \(\text{ImpObvs}\) in a trustworthy host where \(H\) commits to \(C\) and a computation of \(\text{ImpObvs}'\) in \(H\) where \(H\) committed \(C'\) and \(C \neq C'\) with respect to the same \(ea\). Based on Claim 3.10 any transaction including \(ea\) that is submitted to \(H\) by \(\text{ImpObvs}\) was produced by \(O\) to reflect a change in \(O\)'s database. The \(ea\) is produced as a result of \(H\)'s commitments. \(H\) signs any result it produces including commitments. \(H\)'s commitments can be authenticated by using \(H\)'s public key. This means that based on the locations of \(\text{ImpObvs}\), \(O\) produces the same \(ea\) for two different commitments \(C\) and \(C'\). Since \(O\) is trustworthy and follows the algorithm correctly this cannot happen.
\end{proof}
\end{comment}

\begin{claim}
\textit{ImpObvs} is \textit{not-revealing}.
\end{claim}

\begin{proof}
\textit{ImpObvs} is \textit{not-revealing} if for every execution of \(\text{ImpObvs}\) and \(H\) in which \(H\) accomplishes information \(K\) and provides a \(C\), there exists a distributed computation of \(O\) and \(H\), where \(H\) accomplishes the same \(K\) and provides the same \(C\). Thus for every set of transactions \(\text{ImpObvs}\) is not revealing no matter where it is relocated on \(O\) or \(H'\). The proof proceeds by induction on the number of transaction blocks, \(i\), between \(O\) and \(H\). Base case \(i = 0\). In a distributed computation, \(O\) starts a computation after it successfully establishes a connection to \(H\). Thus \(H\)'s only knowledge is that \(O\) wants to perform some computation on its data. In a remote computation of \(\text{ImpObvs}\) in \(H'\), then based on Assumption 2.7, \(H\)'s only knowledge is that \(H'\) wants to perform some computation on its data after a connection was successfully established to it. Thus \(K_{\text{distributed}}(0) \equiv K_{\text{ImpObvs}}(0)\). Since \(H\) does not provide any commitment \(c\) in both schemes at this point, it results that \(C_0(\text{distributed}) = C_0(\text{ImpObvs}) = \phi\), and the base case is proved. Assume now that \(\text{ImpObvs}\) is \textit{not-revealing} for \(r - 1\) blocks. Thus, \(K_{\text{distributed}}(r - 1)\) is the current information accomplished by \(H\) and \(C = \{C_1 \ldots C_k\}\) provided by \(H\) during the computation of the first \(r - 1\) blocks in both schemes. Consider the distributed computation of the \(r^{th}\) block, \(B_r = \{cd_r\} \cup \{ir_{r1}\ldots ir_{ri}\}\). In case \(B_m = \{cd_r\}\) then \(cd_r\) is delivered to \(H\). \(H\) returns with a signed \(c_r\) and achieves information \(k_r\) from \(cd_m\). Thus \(C_{i_1...i_r}(\text{distributed} ) = \{C_1 \ldots C_k\} \cup \{C_r(\text{distributed} )\}\) and \(K_{\text{distributed}}(r) = K_{\text{distributed}}(r - 1) \circ k_r\) where \(\circ\) means the implication of \(cd_r\) on \(K_{\text{distributed}}(r - 1)\). If on the other hand \(B_r = \{ir_{r1}\ldots ir_{ri}\}\) then \(O\) sub-

17
mits all the $ir_j$ where $j = r1 \ldots rl$. Thus $K_{\text{distributed}}(r) = K_{\text{distributed}}((r-1) \circ k_{r1} \circ \ldots \circ k_{rl})$. In a remote computation of $ImpObvs$ in $H'$, $ImpObvs$ receives a message that either instruct it to submit a transaction to $H$ or instructs it to move to the autonomous mode. In the first case, $ImpObvs$ delivers $cd_r$ to $H$, and thus, $C_1 = \{C_1 \ldots C_k\} \cup \{C_r\}$. If on the other hand, $ImpObvs$ is executed in autonomous mode then it receives a set of transactions $B_r = \{ir_{r1} \ldots ir_{rl}\}$. The knowledge that $H$ accomplishes from the set of transactions is $K_{ImpObvs}(r) = K_{ImpObvs}(r-1) \circ k_{r1} \circ \ldots \circ k_{rl}$ and $C_{r-1}(ImpObvs) = C_r(ImpObvs)$.

**Theorem 3.15.** $ImpObvs$ is good.


### 3.3 The communication ratio evaluation for ImpObvs

Taking the above into account we now return to the question that was introduced at the beginning of this chapter: I.e. is there a solution that solves the problem with less overhead, without violating the computation integrity? The communication required by $ImpObvs$ for solving a problem depends on the number of times the two kinds of transaction do interleaved during the execution. This happens as a result of the commitments that $O$ delivers to $H$. Thus, we can state the following theorem.

**Theorem 3.16.** For every problem, $P$, that $Obvs$ solves with $C_r(Obvs)$, $ImpObvs$ solves $P$ with $C_r(ImpObvs)$ where $C_r(ImpObvs) \leq C_r(Obvs)$.

**Proof.** Let us assume that $n$ transactions are issued by $O$ during the remote computation of a given problem $P$ by using $Obvs$. Thus, the number of messages streamed between $O$ and $Obvs$ is $n + cObvs$ where $cObvs$ is a constant number of messages used for maintenance reasons such as relocation, etc. Solving the same $P$ by using $ImpObvs$, it is clear that $O$ issues the same transactions. We present the number of messages streamed between $O$ and $ImpObvs$ to be $r + cImpObvs$ where $r + cImpObvs$ provides the same functionality as $cObvs$. Due to the fact that an interleaving of at most $n$ transactions of different kinds are delivered, then $r \equiv n$. If on the other hand the number of the interleaving transaction of kinds is less then $n$ then $r \prec n$. Based on Claim 3.11 $ImpObvs$ is interactive, i.e. for every message it receives and/or transmits it responses/responded with exactly one message. It results that $C_r(ImpObvs) \leq C_r(Obvs)$. 

18
3.3.1 Algorithm ImpObvs - Test Cases

Proving that ImpObvs is good and presents low communication overhead in comparison to Obvs, the immediate question that follows is: “Given a good deterministic m and H, what is the $Comm(O, m, H)$ for solving a problem as a function of the state in H, the state in O and m itself?” In this section we present two real scenarios that can benefit from remote computation. We use the following notation: \{X : Y, Type, Content\} where X defines the message source, Y defines the message target, Type defines the instruction/information conveyed by the message, and Content that defines the payload.

95Median - Test Case A Consider the problem of computing the 95th median element for a given series of numbers. This evaluation is often realized to compute payment between an ISP and a customer. In such scenarios, the ISP samples the traffic during the mount where each sample is of a constant duration interval (say 10 minutes). Then the 95th median sample is selected as the representative traffic and the customer is paying according. In the distributed computation case, O first requests its ISP to provide it with all the samplings, i.e., the full traffic amount for each interval. Then it computes the 95th median sampling, and commits (sends a check) for that amount. The total communication is thus $O(T \times C)$, where $T$ is the number of intervals (there are about 1000 10 minutes interval in a week), and $C$ is the size of a sample of the traffic.

In the remote computation scenario, O sends the mapplet $m$ to the ISP’s machine. There, $m$ computes the 95th median, delivers it to O for an approval, and submits O’s commitment to H. The amount of communication here is $O(C)$ (ignoring the mapplet size), which as pointed out can be a factor of 1000 smaller than the client/server case. Even if O needs (for the record) a proof for the value committed, $m$ can send it the highest 5% values and their intervals data. The communication in this case will still be a factor of 20 less than the client/server case.

Thus, the transactions issued by O to the ISP (H) as follows:
\{getConnNum, connDuration1 \ldots connDurationn, payCommit\}. We assume that the function evaluateSelfExec() of ImpObvs is implemented to evaluate the 95th median component. The messages flow between O and ImpObvs are:

\{m : O, Relocate, HsignedAddress\},
\{O : mestablishConnectionHost, TargetAddress\},

19
\{m : O, Connect, {}\}
\{O : m, StartCalculation, getConnectionNumber, connectionDuration_1 \ldots connectionDuration_{100}\}
\{m : O, FinalResult, calculated95Median\}
\{O : m, DeliverMsg2Host, payCommit\}
\{m : O, Result, H SatisfactoryCommitment\}
\{O : m, terminateConnection, {}\}. Thus, \(\text{Comm}(O, 95\text{median}, H) = 2*(2*\text{comm}(O, 95\text{median}, H) + 2)\) where the symbol 2 appeared 3 times: the first appearance, of 2, represents the fact that \text{ImpObvs} is interactive; the second appearance represents the fact that there were two interleaving of transaction exchanged, and the last appearances represents the number of time \text{ImpObvs} provides \(O\) with administration information.

**Find Maximum Number - Test case B**

Another interesting problem is computing the maximum value in a given series of numbers. In this example we use the optional extension code, lines 25-26, of \text{ImpObvs}. We first assume that the function \text{evaluateSelfExec()} implements the required algorithm. Thus, the issued transactions are:
\{getValueName, getValue_1 \ldots getValue_{getValueNumber}\}. Note that in this example no commitment is transmitted to \(H\) resulting that the number of interleaved kinds of transactions is 1. \text{ImpObvs} transmits to \(H\) these transactions one at the time and each time that it got a number that is the currently biggest number it informs \(O\) on it. Since the intermediate messages \text{ImpObvs} transmits to \(O\) are affected from the permutation of the numbers that \text{ImpObvs} receives from \(H\), we need to provide the probabilistic of the provided permutation. The probability of a given permutation is

\[
\sum_{i=1}^{n!} \frac{\Pr(O, H) \ast \text{Comm}(O, \text{ImpObvs}, H)}{\text{comm}(O, \text{FindMax}, H)}
\]  

(2)

Thus, the communication for solving the FindMax problem by using \text{ImpObvs} is:
\{\{m : O, Relocate, H signedAddress\},
\{O : mestablishConnectionHost, TargetAddress\},
\{m : O, Connect, {}\}
\{O : m, Autonomous, getValueName, getValue_1 \ldots getValue_{getValueNumber}\}
\{m : O, CheckPointState, H SignedLocalMax\}
\{m : O, Check Point\text{State}, HSignedLocalMax\}
\{m : O, FinalResult, HSignedMax\}
\{O : m, terminateConnection, \{\}\}.

Resulting:
\[
Comm_{Average}(O, FindMax, H) = 2*3*Comm(O, FindMax, H) + \sum_{i=1}^{n} \Pr[O, H] \cdot Comm(O, ImpObvs, H)
\]

3.4 Remarks and conclusions

In this section we presented an improved algorithm for a secured remote computation. As already mentioned at the beginning of this section, relocating a mapplet in $H'$ where the security domain of $H'$ is different then the security domain of $H$ provide us with the following advantages: The possibility that $H'$ influences a mapplet computation to favor $H$ needs is reduced. It is also clear that $H$ cannot tamper with the mapplet data and that $H'$ cannot modify any of the transactions resides in a mapplet since they are signed by $O$.

Nevertheless, this scheme does not assure us that $H'$ will not tamper with the autonomous computation of ImpObvs and that $H'$ will not accomplish any knowledge from the computation. In addition the scheme increases the probability of crashes of either $H$ or $H'$ causing the lost of a computation results.

Based on these considerations, the main interesting issue can be presented as follows: “Is there a resilient scheme that assures a good remote computation and is not based on the black-box Assumption?”

4 A Distributed Scheme of Secured Remote Computation

The simple remote computation scheme presented above is based on the fact that the origin host is trustworthy and thus it can be used as a reference point during the mapplet computation. This scheme requires a steady connection between the origin host and the mapplet. This requirement becomes problematic if the mapplet does not carry its computation close to its origin host, when the origin host does not always remains connected to the network or when the origin host would

\footnote{If it is known that $H'$ is trustworthy then the black-box Assumption can be eased.}
like to remain anonymous to $H$. Other issues to be considered are networks problems, such as nodes and links crashes.

![Figure 5: Multi-Mapplet Distributed Computation](image)

In this section we present a more robust and resilient scheme that is based on fault tolerant and cryptographic techniques. The main idea is to launch a set of mapplets, referred to as computational set, to perform the required task. The computational set includes one mapplet, named $DistribObvs$, that its functionality resembles the above $Obvs$ mapplet, and several auxiliary mapplets, named chaperons, that control $DistribObvs$ execution and replace the origin host as a reference point during the execution (see figure 5). The chaperon’s behavior is interactive (see Definition 3.5). We assume that the computation of a mapplet can be described by a set of predefined transactions. i.e. according to $H'$s response a new transaction out of the predefined set is chosen.

Thus, the chaperons control $DistribObvs$’s execution by computing the next transaction to be submitted to $H$. Every transaction is considered a secret and all the chaperons, included within a given computational set, hold shares of all the possible secrets (transactions). This is done by using the $(k,n)$ secret sharing threshold scheme presented by Shamir in [Sha79].

The scheme computation is carried as follows: All the chaperons, included in a given computational set, execute the exact same algorithm. Thus, based for the same input setting, all the chaperons autonomously evaluate the exact next phase and the same transaction is mutually de-
cided in autonomous fashion. This behaviour holds unless the computation was altered by a mali-
cious/buggy host.

DistribObvs needs at least $k$ shares of the same secret to resolve the signed transaction, which
is then submitted to $H$. $H$ receives the signed transaction, performs its computation and returns
the signed results to DistribObvs which immediately broadcasts it to all other members of the
computational set (i.e. the chaperons).

Theoretically, all the chaperons hold the same database. The database is updated by the chaper-
on’s computation and $H$’s results. At the end of the computation each of the chaperons sends
its database to $O$ (or to an agreed proxy). $O$ receives duplications of the database from all the
chaperons, concludes the execution flows, the results, the commitments, and detects the malicious
host, if there was one, and so on.

The remainder of this section is organized as follows. We first present the cryptographic tech-
nique we use and the additional assumptions needed for the distributed remote computation assurance. Next, we provide a formulation of the shared secrets, and present the distributed algorithm
of the computational set members and prove that it is good. Then we prove that the distributed
scheme works correctly, even when no knowledge is available about the number of the trustwor-
thy hosts with probability that depends on a relation between the computational set size and the
transaction set size. Finally we evaluate the overhead complexity.

4.1 $$(k, n)$$ Secret Sharing Threshold Scheme

The fault tolerant scheme presented in this section makes use of the cryptographic $(k, n)$ secret-
sharing threshold scheme that was introduced by Shamir in [Sha79]. A secret $D$ is divided into $n$
shares $D_1 \ldots D_n$ in such a way that:

- Knowledge of any $k$ or more $D_i$ pieces makes $D$ easily computed.

- Knowledge of any $k - 1$ or less $D_i$ shares leaves $D$ completely undetermined (in the sense
  that all possible values are equally alike).

As described in [Sha79], a random polynomial function $f(x)$ over $Z_p$ is used in the following
way. The degree of $f(x)$ is at the most $k - 1$, and it satisfies $f(0) = D$. Due to the fact that there
is exactly one polynomial of degree at most $k - 1$ satisfying $f(i) = D_i$ for all $k$ values of $i$, the
\((k \cdot n)\) secret sharing threshold scheme is satisfied. Any \(k\) values enable the interpolation of \(f(x)\) and the resolution of \(f(x)\) roots including the free component \(D\). The full details of the scheme is outside of the scope of this paper and the interested reader is referred to [Sha79].

4.2 Assumptions for the Secure Distributed Scheme.

These additional assumptions are used for proving the correctness of the distributed remote computation scheme. The last one is only used for the probabilistic analysis.

Assumption 4.1. Given any set of \(n\) different hosts, at least \(k\) hosts are trustworthy.

This is a reasonable assumption for appropriate values of \(k\) and \(n\). Later on we evaluate the need for this assumption by using probabilistic approach.

Assumption 4.2. The execution of all the chaperons in the computational set is synchronized.

This can be achieved by using common techniques of distributed computation.

Assumption 4.3. The involved hosts do not cooperate with one another.

This assumption is made to disable the case where \(k\) malice hosts will coordinate the execution of the chaperons. To make this requirement even more restrictive we can require that each chaperon relocate itself on a different security administrative domain. If on the other hand a foreign administrative domain is known to be trustworthy then all chaperons may relocate themselves on it, thus resulting in a configuration that is equivalent to the simple remote computation scheme presented in Section 3.

4.3 A Secured Distribute Remote Computing Scheme

In this section we formally define the scheme, describe the algorithm and prove its correctness. When a local computation on a remote host \(H\) is to be performed, a computational set = \(\{\text{DistribObvs, Chaperon}_1 \ldots \text{Chaperon}_n\}\) is launched. Each of the computational set members relocates itself respectively to one host \(\in \{H, H_1 \ldots H_n\}\) where \(H\) is the host where the computation is to be carried locally and \(H_i\) are \(H\)’s close neighbors (see figure 5). \text{DistribObvs} has the same functionality as the mapplet \text{Obvs} that was introduced in section 3 above, with the difference,
that at each computation phase, *DistribObvs* collects at least *k* authentic messages and resolves the shared secret before it continues the execution.

The computation of the distributed scheme continues exactly as described above until the computation is successfully completed or abnormally stopped. Here it cannot be taken for granted that *H*ₙ for *i* = 1 . . . *n* are trustworthy. We only assume that Assumption 4.1 and Assumption 4.2 hold. We do not need the *black-box* Assumption and the chaperons are executed as clear-text.

We first present the formulation of the shared secret distributed among the chaperons. Next, we detail how *DistribObvs* resolves the signed transaction. Next we present the algorithms for *DistribObvs* and for the chaperons in Figure 7 and Figure 8 respectively. We prove that the scheme is indeed *good* and finally we evaluate the probability of the scheme when we replace Assumption 4.3 with Assumption 4.1.

### 4.3.1 Shared Secret Formulation

The predefined transactions set, *T*ₙₙ, is the set of *k* indexed transactions that streams between *O* and *H* for solving a given problem by using a remote computation scheme. Every chaperon receives a permutation of this set and an algorithm that accounts the permutation issue into consideration.

We first assume the existence of an asymmetric crypto-system in which *α* is the private key and *β* is the public one. *Sign* and *Ver* are efficient algorithms for a signature and verification respectively. *Y* is a signature that is verified by *Ver*(*X*, *Y*, *β*) = true iff *Y* = *Sign*(*X*, *α*) (W.H.P). If needed *α* and *β* will be indexed by the originator (either *O* or *H*).

Having the transaction set *T*ₙₙ = {*t*₁ . . . *t*ₙ} we define *ST* = {(,*t*ᵢ, *Sign*(*t*ᵢ, *α*))}. For a secret *X* we define the share of the *j*ᵗʰ element in the (*k*, *n*) threshold scheme to be *SH*ᵣ *ₙ*(*X*). If *k* and *n* are fixed we omit it an use *SH*ᵣ *ₙ*(*X*) to be the *j*ᵗʰ share of the secret *X*. We define *D* to be an *n* × *m* matrix in which *D*ᵢⱼ = *SH*ᵣ *ₙ*(*st*ᵢ) where *st*ᵢ ∈ *ST* and *SD* to be an *n* × *m* matrix in which *SD*ᵢⱼ = (*i*, *D*ᵢⱼ, *Sign*(*i*, *D*ᵢⱼ, *α*)).

The honest dealer, *O* (i.e. the originator), computes *SD* off-line and distributes *SD* among all chaperons, where a *j*ᵗʰ chaperon gets a permutation of the *j*ᵗʰ column of *SD* (its shares of all the secrets *T*).
4.3.2 Secret Shares Construction

Based on the above scheme DistribObvs is designed to receive messages with the form \(((i, D_{ij}), \gamma)\) where \(\gamma\) is \(O\)'s legal signature of \((i, D_{ij})\). It is clear that DistribObvs can ignore any message that does not have the above form or in which the component \(\gamma\) is not a legal signature of \(O\). Thus, DistribObvs receives only authentic messages that were not forged by any unauthorized entity based on Assumption 2.8. By then DistribObvs flushes all messages where their first components do not equal the \(k\) majority. Thus at the end of the message collection phase, DistribObvs holds a set of authentic shares of the form \[\{(i, D_{ij})\}\] where \[\|\{(i, D_{ij})\}\| \geq k\], based on Assumption 4.1.

This set of legal signed shares enables DistribObvs to resolve the secret which has a form of \((t, Sign(t))\). DistribObvs submits the signed transaction to \(H\), and enables it to verify the authenticity of the provided signature over \(t\) by using the public key \(O\). This assures \(H\) that every commitment provided to it by this scheme cannot repudiate by \(O\) at future time. The algorithm for the construction of a transaction is detailed in Figure 6 and it is provided by this paper for aiding the reader.

```c
transaction *ConstructTransaction (n, k, Verification Ko)
    transactions Transaction[n];
    while (SetTimeout(), i < n) 
        M = receiveMsg() where msgPayload = ((j, Data), SignVal);
        if (Ver ((j, Data), SignVal, Ko))
            Transaction [i] = (j, Data);
        if (i >= k) // receives at least k different shares
            j = majority(ind) where (ind, *) \in Transaction[n];
            \{S\} = \{S\} \cup Transaction and elm = (j, Data);
            delete all (ind, *) where ind \neq j;
            if (\|\{S\}\| \geq k) // receives at least k different good shares
                SignedTransaction = resolvePolynomialZeros(S, k);
            return (SignedTransaction);
        handleProblem(); // In case receive less then k good shares
} // of ConstructTransaction
```

Figure 6: The algorithm for resolving a \((k, n)\) transaction sharing
4.3.3 *DistribObvs* and Chaperon algorithms

In this section we prove that the remote distributed scheme is *good*. We also detail the algorithms for the mapplets, *DistribObvs* and the chaperons, included within a computational set. Figures 7 and 8 respectively.

```
mapplet DistribObvs()
    int ChSession[n];
    CurrentHostAddr ← getHostAddress();
    //Finds the chaperons locations, opens a session to all and notifies the current host address
    ChSession ← Resolves&& OpenConct2ComputationalSetLocations();
    DistributeMsg2Chaperons(ChSession, CurrentHostAddr);
    while (TRUE)
        q ← ConstructTransaction (n, k, Verification Ko);
        if (q = (Terminate, _ _))
            closeChaperonsConnections(ChSession);
            selfTerminate();
        SubmitAPI(q);
        SignedRes ← ReceiveResultsfromH();
        DistributeMsg2Chaperons(ChSession, SignedRes);
    mappletEnd DistribObvs
```

Figure 7: DistribObvs - The Distributed Obvious Algorithm

**Claim 4.4.** *The distributed scheme execution results are equivalent to the results of the simple remote computation scheme.*

**Proof.** Based on the remote distributed scheme all the chaperons execute the exact same algorithm and start with the same data image. Based on 4.1 at least \( k \) hosts are trustworthy thus the execution of the chaperons is authentic. Based on Assumption 4.2 and Assumption 4.1 at least \( k \) chaperons will jointly compute the algorithm and produce the expected message to *DistribObvs*. Based on the \((k, n)\) secret sharing scheme \( k \) or more shares enable the resolution of a secret. Thus, *DistribObvs* can resolve the correct transaction. Since the transactions are signed by \( O \) no host can forge them, including \( H \). Thus, the distributed remote computation scheme equals the simple remote scheme.

**Theorem 4.5.** *The computational set execution is good.*
mapplet chaperon
if not (relocated at required host)
    migrate to required host (HostAddress);
sel fTerminate();
Home ← openConnection(HomeAddr);
DistribObvs ← listen(DistribObvsAddr); //wait DistribObvs to open connection
q ← startComputation;
while (TRUE)
    M ← rcvMsg(DistribObvs) where M = (Action, Data);
    if (Action = Terminate)
        CloseConnection(DistribObvs);
        sendMsg(Home, (MyHostAddr, HomeAddr, DBimage)); //send DB image to home
        CloseConnection(Home);
        selfTerminate();
    q ← EvaluateAndDecidesNextTransaction (Data, {Predefined Transactions share set});
    M ← (MyHostAddr, DistribObvsAddr, q);
    sendMsg(DistribObvs, M);
mappletEnd chaperon

Figure 8: The Chaperon Algorithm

Proof. Results immediately from Claim 4.4 and theorem 3.1. □

4.3.4 Participants Hosts Discussion

The interesting issue that is addressed in this section discusses the implications of the information each of the participant hosts archived during the computation on the security of the remote computation.

First it is clear, that on one hand, the foreign host \( H \) understands the transactions submitted to it. On the other hand, it is also clear that it cannot influences the chaperons computation nor can it tamper with their databases.

By assuming that each of the hosts \( H_i \) recognizes the computation purpose and holds the code and data of its chaperon can the computation be forged?

Based on Assumption 4.1, at least \( k \) of the hosts are trustworthy. This means that even if \( n - k \) hosts are malicious the computation cannot be forged. Furthermore, at the end of the computation, \( O \) receives at least \( k \) authentic database images that includes all \( H \) commitments and the external effects shares. Thus \( O \) can resolve its delegate commitments to \( H, H' \)’s commitments,
and the exact point where any malicious hosts tampered with the computation.

4.3.5 Loose Distributed Computation Scheme

The distributed scheme presented in section 4 was proven to be good when at least \(k\) hosts are trustworthy. This requirement seems to be hard to verify since hostile entities do not usually declare themselves as such.

Thus the remaining interesting question is: “If there is no knowledge on the number of the participating trustworthy hosts can the scheme computation be assured with some certainty factor?”

Claim 4.6. For any certainty factor \(\gamma\) there exists a distributed remote computing scheme for which the probability of a “faulty computation” is smaller than \(\gamma\).

Proof. Let's assume that the scheme is insecure for some given algorithm. It’s clear that a false computation occurs when \(H\) can present at least one authentic signed query that cannot be constructed from the chaperon’s database.

Since the scheme requires that any transaction presented by \(H\) will be signed by \(O\), and since \(O\) signs each of its transactions by using its private key, and based on Assumption 2.8 all the hosts are polynomial time bounded; hence neither \(H\) nor any of the participant hosts can forge \(O\)’s signature.

This fact also eliminates the case where \(k\) cooperative malicious hosts carefully select a false transaction, forge the signature and submit it to \(H\) as a legal one. Thus it implies that only \(DistribObvs\) submitted the faulty signed query \(t_{fs}\), while according to the computational set correct execution \(t_s\) should have been submitted, where \(t_s\) and \(t_{fs}\) are two legal transactions of \(O\).

\(DistribObvs\) is basically a simple module. It does not hold a transaction set in its data segment nor can it produce it by its code. Either way it cannot forge \(O\)’s signature based on Assumption 2.8.

The \((k, n)\) secret sharing scheme assures that at least \(k\) buggy / malicious hosts caused \(DistribObvs\) to submit \(t_{fs}\) to \(H\). Based on Assumption 4.3 the participant hosts do not coordinate with each other, and further more cannot cooperatively decide on some approach to forge the computation. This is due to the fact that based on the shares formulation each of the chaperons holds some permutation of its shares.

As a result the \(k\) buggy / malicious hosts autonomously select the \(k\) appropriate shares for producing \(t_{fs}\) with a unified probability.
Assuming the existence of \( m \) different transactions, and \( n < 2k \) active hosts, we get that the probability of having such a situation equals to:

\[
P[t_{fs}] = \frac{1}{m^k} \binom{n}{k} < \frac{1}{m^k} (2k)! = \frac{1}{m^k} \frac{2k(2k-1) \ldots (k+1)}{k!} = \]

\[
= \frac{1}{m^k k!} \prod_{i=1}^{k} \frac{k+i}{i} = \frac{1}{k!} \prod_{i=1}^{k} \frac{k+i}{m}.
\]

Choosing \( ck \leq m \) defines the probability of such a situation to be:

\[
P[t_{fs}] = \frac{1}{k!} \prod_{i=1}^{k} \frac{1}{m} k+i \leq \frac{1}{k!} \prod_{i=1}^{k} \frac{1}{c} k+i \leq \frac{1}{k!} \prod_{i=1}^{k} \frac{1}{c} (1 + \frac{i}{k}) < \frac{1}{k!} \left( \frac{2}{c} \right)^k.
\]

Thus, setting the relation between \( m \) and \( k \) to be a constant \( c \), defines the probability of such a situation. Choosing a large \( c \) causes the probability to be as small as required, and by thus defining the required certainty factor.

\[\square\]

### 4.4 Communication Overhead Evaluation

The overhead that is required for a given computational set and a given \( H \) is composed of three phases:

1. **Launching** The computational set to the participant hosts.

2. **Required Communication** for \( m \) queries and results.

3. **Transmission of the chaperons database image** to \( O \) or a trusted proxy.

1. Let \( Dist_{mean}(H, H_s) \) be the mean distance between \( H \) and \( H_s \).

2. Let \( Comm_{m, mean} \) be the required communication for sending one message for \( Dist_{mean}(H, H_s) \).

3. Let \( Chap_{code}(x), Chap_{data}(x), Chap_{Respec}(x) \) be the chaperons code, data, and results segment size respectively.

Thus, the complexity of the computation is involved with:

- **Launching the computational set:** \( Dist_{mean}(O, H)(n*Chap_{code, data, Respec}(x)) + Dist(O, H)*DistribObvs_{code} \). 

30
For $m$ queries: $2 \times n \times m \times \text{Comm}_{mean}(H_{chap}, H)$ since the size of every share of a secret equals the secret size.

- Sending the results to $O$: $\text{Dist}_{mean}(H_{chap}, O) \times \text{Chap}_{resSpec}(x)$

The required communication overhead is the sum of the above three components. In case the number of the chaperons is small then the communication overhead is reduced. In addition it is clear that there is a trade-off between the number of the chaperons and the provided security as discussed in section 4.3.5.

## 5 Related Work

Protection of the mapplet’s computation is the hardest computational problem in the active system paradigm. Figure 9 presents the different classes of the different approaches.

![Diagram](image)

**Figure 9: Mapplet Protection Models Classifications (KBC00)**

Prevention of mapplet computation tampering is provided *a priori* during mapplet computation and was deemed as an impossible task until Sanders and Tschudin [ST98] first realized that software-based solutions is feasible by using cryptographic methods. [ST98] presented a technique called Computation with Encrypted Function, (CEF), where the mapplet code is encrypted before launching it to be executed at the host. In this way, the host can compute any function, that is representable by a polynomial on its input without being able to tamper with it. Thus the host executes the mapplet as a black-box. This approach was later improved other papers that further generalized the scheme to any arbitrary function that can be represented by a polynomial-size depth [SYY99, CCKM00]. Nevertheless, all these solutions address only the secure evaluation of functions where the final result is produced only to the originator.
Thus, it is not hard to see that such a behaviour is in fact inadequate for a more general behaviour of a computation. Furthermore, in [ACJK01] Algesheimer at al. stated: “(non-interactive)$^6$ secure mobile computing schemes do not exist. In particular, any scheme in which some host is to learn information that depends on the mapplet’s current state cannot be secure”. The suggested solution was to base on a tamper-proof hardware as a trusted entity for such computation behaviour.

6 Conclusion and Future Work

In this paper we investigated several schemes for dealing with assuring mapplets computation in a possible hostile environment without the need of special safe hardware. In this investigation we focused on realistic goals, which are feasible to implement. We do not require that the foreign host to be totally trustworthy, but we seek for a solution that provides us with the same assurance as if the computation was carried using a traditional client/server model.

The presented dynamic scheme is based on distribution of the algorithm on different hosts and uses the transactions as shared secrets by using the $(k, n)$ secret-sharing threshold scheme.

As we showed, the presented scheme supports correct and true computation of mapplets in the sense that it provides the same security assurance as the traditional client/server model while not restricting any of the active systems paradigm, since it is executed as clear-text. This scheme also assures that the information gained by the foreign hosts during the computation is no more than the information gained by distributed computation. The scheme disables situations where none of the involved parties will repudiate their commitments in future time.

The scheme also enables the detection of faulty and erroneous behavior of malicious hosts during execution and not after the computation terminated, thus enabling flushing erroneous results and stopping the mobile applet computation. The scheme enables the detection of any of the malicious hosts that participated in the computation. None of the current models and solutions presented until today support all these abilities. Current models either limit and restrict the active paradigm or are not feasible nowadays.

The distributed scheme provides all these abilities if at least $k$ hosts are known to be trustworthy. In case this information is not available, the paper provides well specified estimation for

$^6$By non-interactive we mean - with the originator or a trustworthy host.
assuring the scheme computation by defining the relation between the number of the transactions and the participant hosts. By using this definition the scheme computation is assured with any required certainty factor.

The main contribution of this paper is the rigorous study of the tradeoff between the security and communication in remote computation. This formal study allows us to take the first step toward understanding the problem, indicating the significance of the various cryptographic assumptions, and the as a results the development of improved remote computation schemes.

Future investigation will be focused both on specific solutions for practical problems and on the ability to react to dynamic situations encountered during the computation with the same security assurance.

References


