ISSN: 2394-7659
IMPACT FACTOR- 3.775

International Journal of Engineering Researches and Management Studies

AN ANALYSIS AND PROPOSAL ON SECURE FAILURE DETECTION AND CONSENSUS IN TRUSTED & UNTRUSTED-PALS

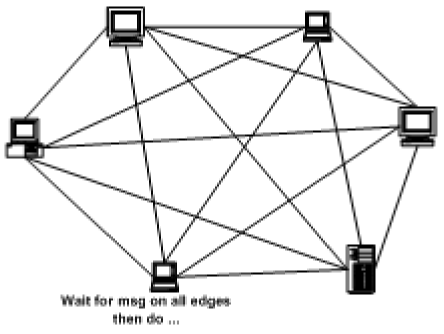
Mr.S.Samson Dinakaran*[1] & Dr.M.Devapriya[2]
*[1]Assistant Professor/Department of CS, VLB Janakiammal College of Arts & Science, Coimbatore, Tamil Nadu, India
[2]Assistant Professor/Department of CS, Government Arts College, Coimbatore, Tamil Nadu, India

ABSTRACT
In this work we present a modular redesign of TrustedPals, a smart card-based security framework for solving Secure Multiparty Computation (SMC). We explore how to make Trusted Pals applicable in environments with less synchrony and show how it can be used to solve asynchronous SMC. The proposed system to explore how to make TrustedPals applicable in environments with less synchrony. In this work, instead of correct processes, we will consider well-connected processes, those processes which are able to compute and communicate without omissions with a majority of processes.

1. INTRODUCTION

In this paper we present limitations of smart card-based security framework for solving Secure Multiparty Computation (SMC). Originally, TrustedPals assumed a synchronous network setting and allowed to reduce SMC to the problem of fault-tolerant consensus among smart cards. We explore how to make Trusted Pals applicable in environments with less synchrony and show how it can be used to solve asynchronous SMC. Within the redesign we investigate the problem of solving consensus in a general omission failure model augmented with failure detectors. The integration of failure detection and consensus into the TrustedPals framework uses tools from privacy enhancing techniques such as message padding and dummy traffic.The proposed system to explore how to make TrustedPals applicable in environments with less synchrony. More precisely, we show how to solve the asynchronous version of SMC using asynchronous synchronization algorithms inspired by recent results in fault-tolerant distributed computing. We use an asynchronous consensus algorithm and encapsulate timing assumptions within a device known as a failure detector. The concept of a failure detector has been investigated in quite some detail in systems with merely crash faults. In such systems, correct processes must eventually permanently suspect crashed processes. There is very little work on failure detection and consensus in message omission environments. In fact, it is not clear what a sensible definition of a failure detector is in such environments because the notion of a correct process can have several different meanings. In this work, instead of correct processes, we will consider well-connected processes, those processes which are able to compute and communicate without omissions with a majority of processes.

2. EXISTING SYSTEM & ITS LIMITATIONS



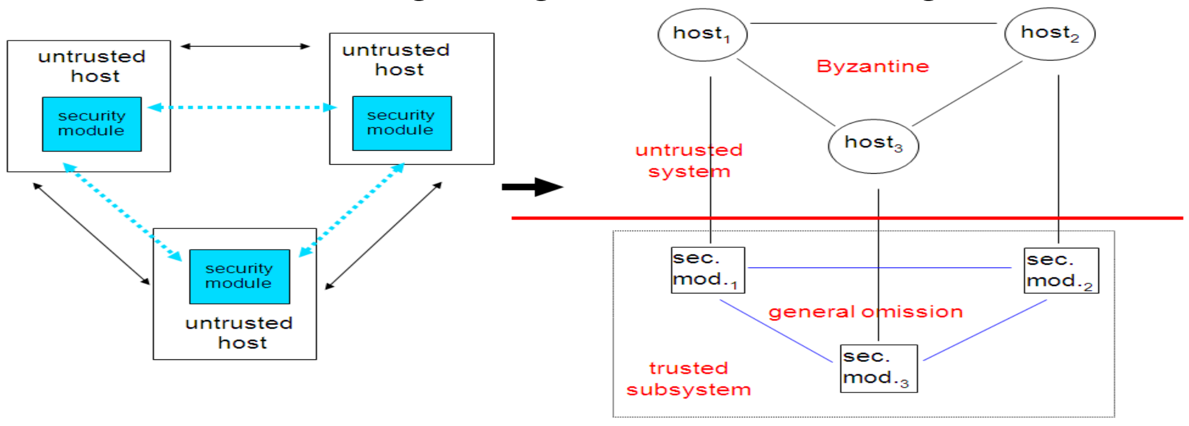
In existing system a set of parties who wish to correctly compute some common function F of their local inputs, while keeping their local data as private as possible, but who do not trust each other, nor the channels by which they communicate. This is the problem of Secure Multiparty Computation (SMC).In "distributed computing" a number of networked players carry out a joint computation of a function on their inputs. The aim of *secure multiparty computation* (or simply, multiparty computation), in contrast, is to enable players to carry out distributed computing tasks on their private information while under attack by an external entity ("the adversary") and/or by a subset of malicious players ("the colluding players"). The purpose of the attack is to learn the private information of non-colluding, honest players or to cause the computation to be incorrect. As a result, there are two important requirements of a multiparty computation protocol: privacy and correctness. SMC is a very general security problem, i.e., it can be used to solve various real-life problems such as distributed voting, private bidding and online auctions, sharing of signature or decryption functions and so on. Unfortunately, solving SMC is—without extra assumptions—very expensive in terms of communication (number of messages), resilience (amount of redundancy), and time (number of synchronous rounds).TrustedPals is a smart card-based security framework for solving SMC which allows much more efficient solutions to the problem. Conceptually, TrustedPals considers a distributed system in which processes are locally equipped with tamper-proof security modules. In practice, processes are implemented as a Java desktop application and security modules are realized using Java Card Technology enabled smart cards, tamper-proof Subscriber Identity Modules (SIM) like those used in mobile phones, or storage cards with built-in tamper-proof processing devices. Roughly speaking, solving SMC among processes is achieved by having security modules jointly simulate a Trusted Third Party (TTP), as we now explain. To solve SMC in the TrustedPals framework, the function F is coded as a Java function and is distributed within the network in an initial setup phase. Then, processes hand their input value to their security module and the framework accomplishes the secure distribution of the input values. Finally, all security modules compute F and return the result to their process. The network of security modules sets up confidential and authenticated channels between each other and operates as a secure overlay within the distribution phase. Roughly speaking, within this secure overlay arbitrary and malicious behavior of an attacker is reduced to process crashes and message omissions. TrustedPals therefore allows reducing the security problem of SMC to a problem of fault-tolerant synchronization, an area which has a long research tradition in fault-tolerant distributed computing (see, for example, Lynch [6]). However, solving the synchronization problem alone is not trivial, especially since we investigate it under message omission failures, a failure scenario which is rather unusual. Furthermore, the reduction from security to fault-tolerance creates a new set of validation obligations regarding the integration of a fault-tolerant algorithm into a secure system, which is also far from being trivial. The initial definition of TrustedPals and its implementation assumed a synchronous network setting, i.e., a setting in which all important timing parameters of the network are bounded and known. This makes TrustedPals sensitive to unforeseen variations in network delay and therefore not very suitable for deployment in networks like the Internet.

Disadvantages

- Highly complex to control *problems*
- Security problem
- Very expensive in terms of communication.
- Message transfer for security may increase the payload so the overall process is denied.

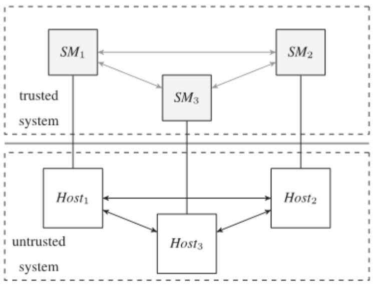
3. PROPOSED SYSTEM & ITS FEATURES

The Proposed system we present a modular redesign of TrustedPals using consensus and failure detection as modules. More specifically, we make the following technical contributions**:**



- We show how to solve asynchronous Secure Multiparty Computation by implementing TrustedPals in asynchronous systems with a (weak) failure detector. We do this by reducing the problem of SMC to the problem of uniform consensus in omission failure environments. As a corollary we show that in systems with security modules and weak timing assumptions the resilience of asynchronous SMC can be improved from a two-thirds majority to a simple majority of benign processes.
- We propose a new definition of connectedness in omission environments. Informally, a process is in connected if it does not crash and, despite omissions, receives either directly or indirectly all messages that a majority of processes sends to it. Similarly, a process is out-connected if it does not crash and all messages it sends are received by a majority of processes. We also consider well-connected processes, which are those processes that are both in-connected and out-connected
- We integrate failure detection and consensus securely in TrustedPals by employing message padding and dummy traffic, tools known from the area of privacy enhancing techniques.

4. UNTRUSTED AND TRUSTED SYSTEM



We formalize the system assumptions within a hybrid model, i.e., the model are divided into two parts .The lower part consists of n processes which represent the untrusted hosts. The upper part equally consists of n processes which represent the security modules. Due to the lack of mutual trust among untrusted hosts, we call the former part the untrusted system. Since the security modules trust each other we call the latter part the trusted system. The processes in the untrusted system (i.e., the hosts) execute (possibly untrustworthy) user applications like e-banking or e-voting programs. Because of the untrustworthy nature of these processes, they use the trusted system as a subsystem to solve the involved security problems. The trusted system consists of software running inside the security modules. This software must have been certified by some accepted authority. It is not possible for the user to install arbitrary software on the trusted system. The tamper-proof nature of the trusted processes allows protecting stored and transmitted information even from the Untrusted processes on which they reside. The authority can be an independent service provider (like a network operator) and is only necessary within the bootstrap phase of the system, not during any operational phases (like running the SMC algorithms).Formally, the connection between the Untrusted and trusted system is achieved by associating each process in the Untrusted system (i.e., each host) with exactly one process in the trusted system (i.e., a security module) and vice versa. Hence, every Untrusted process has a “trusted pal” (an associated trusted process). Since host and security module reside on the same physical machine, we assume that for each association there is a bidirectional eventually timely and secure communication channel, e.g., implemented by shared variables or message passing communication in the host operating system.

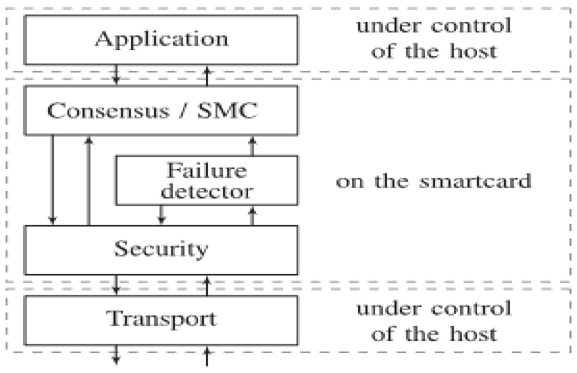
Untrusted System: Assumptions
Within the Untrusted system each pair of hosts is connected by a pair of unidirectional communication channels, one in each direction. We assume that there is a minimal set of reliable channels in the system. The rest of the channels can be lossy. Recall that every message sent through a reliable channel is eventually delivered at the destination. We assume no particular ordering relation on channels.

- Failure Model
- Timing

Trusted System: Assumptions
The trusted system can be considered as an overlay network—a network that is built on top of another network—over the Untrusted system. Nodes in the overlay network can be thought of as being connected by virtual or logical channels. In practice, for example, smart cards could form the overlay network which runs on top of the Internet modeled by the untrusted processes. In the trusted system, each process has also an outgoing and an incoming communication channel with every other process.

- Trust Model
- Timing
- Failure Model

5. ARCHITECTURE OF PROPOSED SYSTEM & PROPOSED FEATURES



Advantages

- Eliminate the Secure Multiparty Computation (SMC) by using smart card-based security.
- Omission of failure in environments.
- No security issues.
- Very little work on failure detection.
- Payload in the Network for security issues can be reduced.

References

1. *A.C.-C. Yao, “Protocols for Secure Computations (Extended Abstract),” Proc. IEEE 23rd Symp. Foundations of Computer Science (FOCS), pp. 160-164, 1982.*
2. *M. Fort, F.C. Freiling, L.D. Penso, Z. Benenson, and D. Kesdogan, “Trustedpals: Secure Multiparty Computation Implemented with Smart Cards,” Proc. 11th European Symp. Research in Computer Security (ESORICS), pp. 34-48, 2006.*
3. *Z. Chen, Java Card Technology for Smart Cards: Architecture and Programmer’s Guide. Addison-Wesley Longman Publishing Co., Inc., 2000.*

4. N. Leavitt, "Will Proposed Standard Make Mobile Phones More Secure?," Computer, vol. 38, no. 12, pp. 20-22, Dec. 2005.
5. Certgate GmbH, "Certgate Smart Card," http://www.certgate.com/web_en/products/smartcardmmc.html, 2008.
6. N.A. Lynch, Distributed Algorithms. Morgan Kaufmann Publishers, Inc., 1996.
7. T.D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," J. ACM, vol. 43, no. 2, pp. 225-267, 1996.
8. F.C. Freiling, R. Guerraoui, and P. Kuznetsov, "The Failure Detector Abstraction," ACM Computing Surveys, vol. 43, no. 2, pp. 1-40, 2011.
9. D. Malkhi and M.K. Reiter, "Unreliable Intrusion Detection in Distributed Computations," Proc. 10th Computer Security Foundations Worksop (CSFW), pp. 116-125, 1997.
10. K.P. Kihlstrom, L.E. Moser, and P.M. Melliar-Smith, "Byzantine Fault Detectors for Solving Con sensus," Computing J., vol. 46, no. 1, pp. 16-35, 2003. ECT