Project N^o: **FP7-284731**

Project Acronym: **UaESMC**

Project Title: **Usable and Efficient Secure Multiparty Computation**

Instrument: **Specific Targeted Research Project**

Scheme: **Information & Communication Technologies**

Future and Emerging Technologies (FET-Open)

Deliverable D4.2.1
Algorithms

Due date of deliverable: 31st January 2014

Actual submission date: 31st January 2014

Start date of the project: **1st February 2012** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **KTH**

Specific Targeted Research Project supported by the 7th Framework Programme of the EC		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Algorithms

This document constitutes deliverable D4.2.1 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

The report presents algorithms developed in workpackage WP4.2.1 and contains an overview of our research results and an appendix. The overview presents the application scenarios and the existing state of the art and summarizes the algorithms developed in the work package. The appendix consists of five technical reports, each one presenting the technical details of the implemented solutions.

The deliverable is an intermediate version of the final deliverable D4.2.2.

List of Authors

Roberto Guanciale (KTH)
Dilian Gurov (KTH)

Contents

Chapter 1

Introduction

This document summarizes deliverable D4.2.1 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

The document present algorithms developed in workpackage WP4.2.1. The main goal of the proposed techniques and algorithms is to effectively apply secure multiparty computation to real applications. With this aim, we take into account realistic networking and resource constraints. The deliverable is an intermediate version of the final deliverable D4.2.2 that is due at the end of the project.

The document is composed of two chapters:

- Chapter 2 provides an overview of our research results. The chapter presents the application scenarios and the existing state-of-the-art and summarizes the algorithms developed in the work package.
- The Appendix is composed of five technical reports that present the technical details of the implemented solutions.

Chapter 2

Overview

This Chapter provides an overview of our research results. The chapter presents the application scenarios and the existing state-of-the-art and summarizes the algorithms developed in the work package D4.2.1. We focus on three main application scenarios:

- Section 2.1 presents a Peer-to-Peer positioning system. The system is a GPS-complementary infrastructure for location-aware applications. Appendix A.1 describes an algorithm to implement the Peer-to-Peer localization based on trilateration and partially homomorphic cryptosystems.

- Section 2.2 presents a cost-aware routing mechanism for autonomous systems. Our main goal is to allow competitive parties to collaborate to achieve cost-effective routing tables. Appendix A.4 describes the implementation of a modified version of the Belman-Ford algorithm using secret sharing, which requires a centralized infrastructure. Appendix A.5 describes a distributed algorithm to compute single source shortest paths that uses SHAREMIND as mathematical blackbox.

- Section 2.3 presents collaborative monitoring. The presented infrastructure enables distributed (and possible unaware) parties to jointly increase the effectiveness of their intrusion detection and debugging infrastructures. In Appendix A.2 we identify the information leaked by distributed monitoring algorithms in different scenarios and show how the general problem can be addressed by privately computing language intersection among subset of involved partners. In Appendix A.3 we present a prototype implementation of privacy-preserving intersection of regular languages. Our algorithm starts from a Deterministic Finite Automaton (DFA) representation of the input languages, computes their product and then minimizes the resulting product automaton, yielding a canonical (and thus non-leaking) representation of the language intersection.

2.1 "P2P" positioning system

We investigated one of the applications proposed in the deliverable Section 3.1.2 of Deliverable D4.1 [9]: a peer to peer positioning system. Location-aware mobile applications rely on two main infrastructures: the Global Positioning System (GPS) is adopted as ubiquitous available service for outdoor car navigation, WiFi coverage is used to increase precision or indoor positioning. In several scenarios, exploiting these approaches is not sufficient. A peer to peer positioning system can support the existing infrastructure, by providing a further measure. In this scenario several mobile peers dynamically discover their positions (e.g. by using the existing infrastructures or the p2pps itself) and act as further information sources for other peers.

A peer Q exploits the application to discover its location $p_Q = (x_Q, y_Q, z_Q)$. We assume the availability of n other peers P_i that know their own positions $p_i = (x_i, y_i, z_i)$. We also assume that it is possible to estimate the distance r_i between the peer Q and each other peer P_i. The distances can be obtained by using several electromagnetic measures, ranging by WiFi beam signals, FM signals and bluethooth. These inputs allow to adopt "trilateration" to compute the unknown position of the peer Q : for each peer P_i, the position p_i and the distance r_i identify a sphere surface:

$$r_i^2 = (x_Q - x_i)^2 + (y_Q - y_i)^2 + (z_Q - z_i)^2$$

The equation corresponding to the peer P_n can be used to linearize the quadratic equation system: the position p_Q can be obtained as $p_Q = s + p_n$ where s is the solution of the linear system of equations $As = b$:

$$A = \begin{pmatrix} x_1 - x_n & y_1 - y_n & z_1 - z_n \\ ... & ... & ... \\ x_{n-1} - x_n & y_{n-1} - y_n & z_{n-1} - z_n \end{pmatrix} \quad \vec{s} = \begin{pmatrix} x_Q - x_n \\ y_Q - y_n \\ z_Q - z_n \end{pmatrix} \quad \vec{b} = \begin{pmatrix} b_1 \\ ... \\ b_{n-1} \end{pmatrix}$$

$$where\ b_i = \frac{r_n^2 - r_i^2 + (x_n - x_i)^2 + (y_n - y_i)^2 + (z_n - z_i)^2}{2}$$

If the resulting system has three linear independent equations (namely there are four peers that know their own positions) and the measures are not affected by errors, then trilateration is reduced to solving the resulting linear system of equations. If the linear system is overdetermined or the measurements are affected by errors, the nearest solution can be discovered by solving the linear least squares problem, estimating the s as the position $\bar{s}$ that fits the "best", in the sense of solving the quadratic minimization problem:

$$\bar{s} = argmin_s(\mid b - As \mid^2)$$

The linear least squares has an unique solution if the columns of A are linearly independent (i.e. there are at least three linearly independent position of the involved peers). The problem can be solved by handling the "normal equations": $(A^T A)s = A^T b$.

In this application we assume that the peer Q knows the distances r_i of the other peers and wants to discover its own position. The other participants know their own positions and must not be able to discover any information on the location of Q.

The general approach [11] to securely compute the p_Q consists in building a circuit that performs "Gauss-elimination". For a system of n equations and n variables, "Gauss-elimination" requires

- $n(n-1)/2$ divisions
- $(2n^3 + 3n^2 - 5n)/6$ multiplications
- $(2n^3 + 3n^2 - 5n)/6$ subtractions

Moreover, in the above equation system, the system coefficients are not public or directly known by any participant: A_{ij} and b_i depend on information known by the participants P_i and P_n. The corresponding "preprocess" circuit requires $3n$ multiplications (to compute the squares of the distances in b) and $(n+3)*n$ additions.

In [6] the authors handle the problem for two party environment when $A = A_1 + A_2$ and $b = b_1 + b_2$. Even if our methodology is inspired by their work, their solution can not directly applied, since b is not a linear combination of vectors know by the participants.

In Appendix A.1 we propose an ad-hoc protocol. We use a partially homomorphic cryptosystems (e.g. Benaloh [2] and Paillier [15]). The algorithm exploits two key ideas:

- usage of an additive homomorfic cryptosystem for public key crypthographic, Q knows the private key and all other participants know the corresponding public key
- computing $A\vec{s} = \vec{b}$ is equivalent to computing $J(A)RR^{-1}\vec{s} = J\vec{b}$, where J and R are two invertible (random) matrices.

2.2 Collaborative cost aware routing

Internet packets are delivered through a complex network of Internet Service Providers (ISPs). Each ISP owns an Autonomous System (AS) that participates in the Border Gateway Protocol (BGP) [16]. The Border Gateway Protocol is exploited to make inter-AS routing decisions, allowing each ISP to independently enforce its own policy rules and to locally compute its preferred intra-AS routing strategy. Several scenarios can take benefits by more collaborative ASs. The number of current (2012) ASs is estimated as 40000.

An autonomous system manages a set of IP blocks. For example, an AS managing a.b.0.0/16 hosts a contiguous block of 65536 IP addresses, ranging from a.b.0.0 to a.b.255.255. An autonomous system can manage more than one IP block.

An AS is a network of routers and hosts (both routers and hosts are identified by IP addresses, but hosts can not forward packets). Internally, an AS exploits a distance vector/link state algorithm to compute the internal routing strategy. To deliver a packet between two internal hosts, only internal AS information are used.

Some routers of the AS are connected with routers of other ASs. These routers are called border gateways. If an AS owns more than one border gateway, it is usually referred as multihomed AS (or domain). The routing protocol used to establish the routing tables of the gateways is BGP (path vector protocol). The routing table allows each gateway to store the "best" next gateway to reach each possible IP block. While the intra-routing algorithms exploits cost metrics to compute the "best path", currently the BGP is cost independent and allows each gateway to select the preferred path using a local policy.

2.2.1 Non-compact inter-AS routing

The network can be represented as a undirected graph $G = (V, E)$, consisting of V vertices and $E \subseteq V \times V$ edges. Since we focus on inter-AS routing, each vertex $v \in V$ represents one border gateway.

An autonomous system is uniquely identified by a name ranged over by $a_1, \dots, a_n$. We use A to refer to the set of all autonomous system names. The function $as : V \to A$ yields the owner of a gateway. We use $gws(a) = \{v \mid as(v) = a\}$ to represent the set of gateways owned by the autonomous system a. An autonomous system owning more than one border gateway is a multihomed system. We assume that there is no multihomed system in the network.

Let W be a function ($W : E \to R^{0+}$) that represents the (non-negative) weights of the edges. We assume that:

- if $e = (v_1, v_2)$ and $v_1 \in gws(a_1), v_2 \in gws(a_2)$ then only a_1 and a_2 know the cost $W(e)$ (these costs represent commercial agreement or inter-AS congestion levels, network delays)
- we consider public the network topology

A path of vertices $p = v_1, \dots, v_n$ is a sequence of vertices such that v_i is adjacent to v_{i+1}. We use $s(p)$ and $t(s)$ to represent the "source" (first) and "target" (last) node of a path respectively. The cost of a path is computed according to the sum of edge weights (even if alternative approaches can require to aggregate the costs using min or max). Let (v_i, v_j) be the edge connecting the vertex v_i to the vertex v_j, the "cost" of a path of n nodes is defined as the sum $w(p) = \sum_{i=1}^{n-1} W(p[i], p[i+1])$.

A routing strategy:

- provides a "protocol" to jointly compute (or update in a dynamic scenario) a local data structure $\Delta_{v \in V}$ (routing information) for each gateway
- provides a function $f(\Delta_v, m) = (v', m')$, that allows a gateway v to locally compute the next gateway to which the packed m must be delivered to reach the destination. Notice that the routing function can allow to rewrite the forwarded packet.
- the routing strategy musts guarantee that packets are delivered only to neighbours gateways: $\forall m. f(\Delta_v, m) = (v', m') \Rightarrow (v, v') \in E$.

- $dts(m) \in V$ represent the destination gateway of the message m
- the path to deliver the packed m from the gateway v_s is $P(v_s, m) = v_s \; if \; v_s = dst(m) \; otherwise \; v_s, P(f(\Delta_v, m))$
- the routing strategy accomplishes the best path if for all message m (destination) and source gateway v_s does not exists a path p such that $s(p) = v_s$, $t(p) = dst(m)$ and $w(p) < w(P(v_s, m))$.

The standard approach is based on routing tables:

- each message m contains the destination gateway (or the destination IP block)
- the computed routing information for each gateway Δ_v is a routing table $RT_v : V \rightarrow p$, namely it is a total function on graph vertices that yields the full gateway path to reach the vertex
- the routing function $f(\Delta_v, m) = (RT_v(dst(m))[0], m)$, namely it returns the next hop of the path stored into the routing table and does not change the message
- The BGB protocol is cost agnostic and does not accomplish best path routes

In Appendix A.4 and Appendix A.5 we present two approaches to compute single target shortest paths. These algorithms outputs for each gateway a single entry of the routing table, which allows to accomplish the most cost-effective delivery of packets target to a single destination. The union of these entries of the gateway routing tables represent a routing three. The algorithms uses two different strategies:

- the algorithm presented in Appendix A.4 implements a modified version of Belman-Ford algorithm [1, 8] in Sharemind. The algorithm allows to compute the routing three using three servers (miners) that adopt secret sharing, requiring a centralized infrastructure
- the algorithm presented in Appendix A.5 implements a modified version of MarlinSegall algorithm [14] in Sharemind. The algorithm allows to compute the routing information by distributing the algorithm on several instances of Sharemind, shared among network neighbors. In the report, we also describe how relaxing the privacy constraints can speed up the computation.

2.2.2 Truthful path allocation

Intuitively, the costs of the graphs presented in Section 2.2.1 represent the damage the gateways incur to exchange a packet (e.g. network congestion). In this network we want to deliver a packet between two gateways and we want to maximize of the *social welfare*, that in this example is to minimize the total damage we cause.

In this settings, one of the party (i.e. an autonomous system) can lie about their own costs, reporting higher values that cause the link to not be allocated in any packet delivery. To prevent this behavior, a payment can be guarantee for incentive the parties (and compensate their costs).

In Appendix A.4 we investigate we investigate a payment scheme that incentives truthful reporting of the party costs and we show how the payment each party has to receive can be computed without compromising their private informations (the network costs).

2.3 Distributed Monitoring and Collaborative Intrusion Detection

2.3.1 Collaborative Intrusion Detection

In collaborative intrusion detection, a number of distributed agents collaborate to identify possible threats and attacks. Attacks are usually of the type one-to-many (as in stealthy scans, where the attacker scans large numbers of hosts simultaneously in search of software vulnerabilities) or many-to-one (as in distributed denial-of-service attacks, where a multitude of compromised hosts overload a target system). Attacks of the first type are particularly difficult to identify in time without collaboration of the targeted agents, since the local events of the individual agents in isolation may not be indicative of an attack.

Components Collaborative intrusion detection typically involves some of the following components:

- *Event logs*: Agents *monitor* local events stored as event logs to detect possible (local) attack steps.
- *Alerts*: A class of self-standing attack steps, such as alerts flagging incoming packets (their volume, port, source distribution patterns) that were denied by a firewall, or viruses (their type, target, and action taken) detected by antivirus software.
- *Attack model*: A model that describes the individual actions (attack steps) of a given attack (scenario) and their possible sequencing. Attack models can be *implicit*, i.e. based on similarity (using data-mining techniques to aggregate and cluster events), and *explicit*, i.e. based on learned or human-provided attack scenarios described in a language that can express logical and temporal constraints on events. Examples for attack description languages are goal trees [10], which can be viewed as the formation trees of regular expressions or process algebraic terms, and LAMBDA [5], a language based on pre- and post-conditions and scenarios.

Activities Collaborative intrusion detection typically involves several of the following activities:

- *Alert correlation*: Local attack steps from different agents are correlated to identify (global sequences of) attack steps pertaining to the same attack.
- *Attack identification*: The so identified global sequences of attack steps are *matched* against the attack model to identify a possible attack and its (global) attack state.
  - *Attack threads*: The match is rarely perfect, since agents don't always detect all relevant attack steps. For this reason, (incomplete) attack threads are identified and weighted. Measures such as step probabilities, thread length and level of completeness can be employed to identify the most likely attacks [10].
  - *Holes*: The (most likely attack) threads are analysed w.r.t their holes (i.e., missing attack steps). These are further investigated (locally) by the agents in an attempt to substantiate or exonerate the attack [10].
- *Attack prediction*: The identified attack threads are used to predict the future steps of the attacker, resulting in alerts to the target agents. These can trigger a raise in the *audit level* of these agent to pro-actively look for data relevant to the anticipated attack steps.
- *Attack model mining*: The observed event sequences can also be used to learn new attack models or refine existing ones.

Architectures Collaborative intrusion detection systems can be based on various architectures, centralizing or distributing the individual activities in various ways.

- *Hierarchical* architectures partition agents into local and global ones. The local ones monitor their local events, while the global ones are responsible for maintaining the attack models. This division determines the flow of information between the two types of agents.
- *Fully distributed* (or peer-to-peer) architectures, where each participant has two function units: a *detection unit* that is responsible for the monitoring of local events, and a *correlation unit* that is part of a distributed correlation scheme.

Privacy Concerns There are various privacy concerns arising from CID:

- Participants may be unwilling to share alerts that contain sensitive information about their network or users. Raw alerts may expose site-private topological information, proprietary content, client relationships, or the site's defensive capabilities and vulnerabilities. According to [12], sensitive fields in typical alert formats are IP addresses (such as Source_IP and Dest_IP) and data contained in Captured_Data and Infected_File.
- Certain attack models may be sensitive, or be used for probe-response attacks as described in [12]: the attacker can attack a particular system and then observe the alerts published by the repository to determine whether the attack has been detected, and if so, how it has been reported.

Data correlation systems that provide strong privacy guarantees for the participants achieve data privacy by means of (partial) data *sanitization* (based for instance on Bloom filters) or by means of SMC. However, data sanitization loses information and thus imposes an unavoidable tradeoff between data privacy and data utility.

In previous work Sepia, a Java library for generic secure multiparty computation [4] based on Shamir secret sharing, implements a number of algorithms (or protocols) that can be used as building blocks for privacy-preserving collaborative network monitoring, such as:

- *Event Correlation*: Reconstruct a given event if and only if at least a given threshold number of agents report the event and the aggregate weight is at least another given threshold weight. Aggregation enables the (early) detection and characterization of attacks spanning multiple domains using data from IDSs, firewalls, and other possible sources.

and network traffic statistics:

- *Vector Addition*: Privacy-preserving addition of r-dimensional vectors.
- *Entropy Computation*: Computation of Tsallis entropy of feature distributions such as IP addresses, port numbers, flow sizes, or host degrees. Has been successfully applied in network anomaly detection.
- *Distinct Count*: Counting privately the number of agents seeing a given event.

For now, we do not envisage to contribute to this line of work.

2.3.2 Distributed Monitoring

Assume a set of distributed agents that keep logs over locally chosen observable events. When things go wrong in the system, the usual question is the one of: "What happened?" Distributed monitoring is the problem of computing the possible (global) executions that are compatible with the (local) logs recorded by the agents. For very large distributed systems, however, the more meaningful problem is not the one of computing a global solution, but the one of computing local views of the solution, in a distributed fashion. In other words the problem is, for each agent, to infer what happened locally, that is, by communicating with the other agents to compute all possible local executions that are (i) locally consistent with the logs, and (ii) globally synchronizable. This problem is known as *modular distributed monitoring.*

An important issue is the mathematical foundation on which to base modular distributed monitoring. Conceptually, it is natural to think in terms of partial orders between events (or event occurrences). In [7] such a theory is developed in terms of products, projections and an efficient representation of partial orders called trellis.

A problem worth investigating is whether and how the theory can be combined with SMC techniques for building a library of operations as building blocks for privacy-preserving modular distributed monitoring, and potentially also for attack identification and attack model mining, for instance using trellis to represent attack models.

Formal Statement of the Problem We formalise the problem as in [7].

Let $\mathcal{L}$ be a language over alphabet L. Then let $\mathbf{proj}_{L'}(\mathcal{L})$ for $L' \subseteq L$ denote the *projection* of $\mathcal{L}$ onto the alphabet L', defined as expected through deleting letters not in L'. Projection can be generalized to arbitrary alphabets as follows: $\pi_{L'}(\mathcal{L}) \overset{\text{def}}{=} \mathbf{proj}_{L'}^{-1}(\mathbf{proj}_{L\cap L'}(\mathcal{L}))$. Notice that if $L' \subseteq L$ then $\pi_{L'}(\mathcal{L}) = \mathbf{proj}_{L'}(\mathcal{L})$.

The *product* of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ over alphabets L_1 and L_2, denoted $\mathcal{L}_1 \times^L \mathcal{L}_2$, is the largest language $\mathcal{L}$ over $L_1 \cup L_2$ such that $\mathbf{proj}_{L_1}(\mathcal{L}) = \mathcal{L}_1$ and $\mathbf{proj}_{L_2}(\mathcal{L}) = \mathcal{L}_2$. In terms of finite automata, this corresponds to a weakly synchronous product[1].

We assume a set I of agents (or sites). For each site $i \in I$ we are given a local alphabet of actions (or events) L_i partitioned into the observed (or recorded) actions $L_{o,i}$ and unobserved actions $L_{u,i}$, a local behaviour (or set of executions) given as a language $\mathcal{L}_i \subseteq L_i^*$, and a local observation (or log) $\omega_i \in L_{o,i}^*$ consistent with $\mathcal{L}_i$, i.e., $\omega_i \in \mathbf{proj}_{L_{o,i}}(\mathcal{L}_i)$.

Let $\mathcal{L} \overset{\text{def}}{=} \times_{i\in I}^{L} \mathcal{L}_i$ and let $\Omega \overset{\text{def}}{=} \times_{i\in I}^{L} \{\omega_i\}$. The product $\mathcal{L} \times^L \Omega$ represents all global executions that are consistent with the logs, and can be equivalently represented by the product $\times_{i\in I}^{L} (\mathcal{L}_i \times^L \{\omega_i\})$, where $\mathcal{L}_i \times^L \{\omega_i\}$ represents all local executions of site i consistent with the respective local log ω_i. The problem of distributed monitoring is formally defined as computing the map:

$$(\omega_i)_{i\in I} \mapsto (\mathbf{proj}_{L_i}(\mathcal{L} \times^L \Omega))_{i\in I} \tag{2.1}$$

In Appendix A.2 we present an algorithm to address distributed monitoring. In particular, we focus on identifying the information leaked in different scenarios and we show how the problem can be addressed by privately computing language intersection if the parties involved are two. In Appendix A.3 we implement privacy preserving intersection of regular languages. Our algorithms start from the Deterministic Finite Automaton representation of the input languages and compute the minimization of the product automata.

2.3.3 Collaborative Intrusion Detection on top of Distributed Monitoring

Attack Identification Assuming an attack model that describes the possible event sequences of an attack scenario, the task is to match the local events of the agents against this model. The privacy requirements are that this match should not leak information between the agents other than the result of the match. The result of the computation can be used by the local agents to discover if a single attacker is coordinating activities that involve the IT infrastructures of the agents.

We formulate a first version of the problem abstractly as follows, using our notation and set-up from Section 2.3.2. The attack model is defined extensionally through a language $\mathcal{L}_A$ over an abstract alphabet L_A of events (attack steps). We assume that the local agents own as private input event logs[2] in the form of strings ω_i, while a dedicated global agent[3] privately owns the attack model $\mathcal{L}_A$. Our task is to decide:

$$\mathcal{L}_A \cap \left(\mathcal{L} \times^L \Omega\right) \neq \emptyset$$

in a privacy-preserving fashion, using SMC.

A variation of the problem is the one of *partial match*, based on some threshold measures on the length of strings and the degree of matching. The partial match should be able to yield a match measure for a log that contains events not expected by the attack model:

- yielding match metric for "imperfect" match allows to identify the "best matching" attack model;
- an unmatched alert should be notified only to the corresponding reporting local agent;

[1] The adequacy of a synchronous product in the context of distributed systems (usually using asynchronous message passing for communication) can be questioned. However, by interpreting $\times^L$ as language *composition* in a more general sense, the proposed formalization remains meaningful.

[2] We assume that an all the event logs correspond to a unique instance of a coordinated attack. Splitting real logs into attack instances is in charge of the correlation activities.

[3] The exact architecture needs to be thought over: is the attack model to be kept by a local agent, and is the attacker itself a local agent? If yes, what are their alphabets?

- unmatched alerts can be used as feedback to the correlation algorithms, which usually exploit data-mining techniques.

An additional task is to identify *holes* as well as possible *next events*. Similarly to partial match, the holes discovered can be used as feedback for the correlation algorithms.

Another variation of the above is a setting where the attack model is shared privately between the local agents. A particular scenario involves the language $\mathcal{L}$ built as the product of the agent languages $\mathcal{L}_i$. In this settings, the agents are interested to keep their languages secret, since these can reveal vulnerabilities and the internal IT infrastructure.

Attack Model Mining In this setting there is no attack model in the beginning. Instead, the task is to compute one from the individual logs of the agents.

We formulate a first version of the problem abstractly as follows. Again, we assume the local agents owning as private input event logs in the form of strings ω_i. The task is to compute in a privacy-preserving fashion the language:

$$\mathcal{L} \times^{L} \Omega$$

Next, the problem can be tied to a particular attack description language, using techniques from data mining or process mining. The latter problem has been studied in detail in [3] (without considering privacy), where a number of process mining techniques are compared in the context of *process trees*, a notion closely related to the goal trees described in [10]. In this setting the learned model need not be exactly the set defined above, but can approximate it, balancing suitably between a number of quality measures. A possible task can be to develop privacy-preserving versions of some of these algorithms.

SMC-based privacy-preserving data mining techniques are developed in [13]. Again, there is the choice of making the learned model known to all agents, making it known to only one global agent, or sharing it privately between the agents.

Bibliography

[1] Richard Bellman. On a routing problem. Technical report, DTIC Document, 1956.

[2] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994.

[3] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Conferences (1)*, pages 305–322, 2012.

[4] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, pages 223–240, 2010.

[5] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Recent Advances in Intrusion Detection*, pages 197–216, 2000.

[6] Wenliang Du and Mikhail J Atallah. Privacy-preserving cooperative scientific computations. In *csfw*, volume 1, page 273. Citeseer, 2001.

[7] Eric Fabre and Albert Benveniste. Partial order techniques for distributed discrete event systems: Why you cannot avoid using them. *Discrete Event Dynamic Systems*, 17(3):355–403, 2007.

[8] Lester Randolph Ford. Network flow theory. 1956.

[9] Roberto Guanciale. Identification of application scenarios, January 2013. UaESMC Deliverable 4.1.

[10] Ming-Yuh Huang, Robert J. Jasper, and Thomas M. Wicks. A large scale distributed intrusion detection framework based on attack strategy analysis. *Computer Networks*, 31(23-24):2465–2475, 1999.

[11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, volume 201, 2011.

[12] Patrick Lincoln, Phillip A. Porras, and Vitaly Shmatikov. Privacy-preserving sharing and correlation of security alerts. In *USENIX Security Symposium*, pages 239–254, 2004.

[13] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR Cryptology ePrint Archive*, 2008:197, 2008.

[14] Philip M. Merlin and Adrian Segall. A failsafe distributed routing protocol. *Communications, IEEE Transactions on*, 27(9):1280–1287, 1979.

[15] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[16] Yakov Rekhter and Tony Li. A border gateway protocol 4 (bgp-4). 1995.