electronics

MDPI

Article

SmartContent—Self-Protected Context-Aware Active Documents for Mobile Environments

Francesco Luca De Angelis * and Giovanna Di Marzo Serugendo

Institute of Services Science, University of Geneva, Route de Drize 7, CH-1227 Carouge, Switzerland; giovanna.dimarzo@unige.ch

* Correspondence: francesco.deangelis@unige.ch; Tel.: +41-223-790-259

Academic Editor: Martin Werner

Received: 19 January 2017; Accepted: 22 February 2017; Published: 4 March 2017

Abstract: Mobile devices make it possible to create, store, access, share or publish personal content on the Internet, anywhere and at anytime. This leads to situations of potential intentional or unintentional misuse of content as well as privacy issues. Recent techniques involving the use of contextual information focus on access of documents stored in clouds, or authentication for secured Web sites. These techniques or more traditional solutions, such as steganography or Digital Rights Management, do not empower the user itself, or data controller in professional settings, with a fine-grained control of the access to or manipulations actions on documents stored on mobile devices, e.g., copying, sharing, etc. In this paper, we propose SmartContent, a novel approach for content protection and privacy. Documents are active and context-aware documents that sense and analyse their current context, e.g., location, noise, neighbouring devices, social network, expiration time, etc. Based on user provided policies, they grant, deny or limit access and manipulation actions, or destroy themselves if necessary. We present the generic model of SmartContent, a concrete architecture and an implementation of a proof-of-concept specifically designed for mobile devices. We deployed it on tablets and showed that a picture dynamically reveals or conceals itself based on sensed context or on changing policies. The implementation leverages the SAPERE middleware specifically developed for context-aware systems.

Keywords: self-protection; security; mobile

1. Introduction

The development of Internet and the diffusion of mobile phones allow anyone to easily publish and share personal content such as images, videos or music. This facility of sharing is accompanied by a major drawback: it makes it difficult or even impossible for individuals to control the propagation or copies of such data. It hinders the protection of privacy: we may want to share our picture on a social media with our own friends, but disagree with the use of this picture by private companies for advertising purposes or refuse that our friends share this picture with friends of their own. There is a need for individuals, citizens or professionals like for instance chief information officers, to empower themselves and take control of data stored under electronic form.

Traditional solutions, such as steganography [1,2], allow identifying copies, but do not prevent viewing or manipulating content. Digital Rights Management techniques [3,4] prevent viewing but rely on proprietary material and do not consider the context or situation related to current access request. Current solutions, involving the use of contextual information, focus on access of documents stored in clouds, or authentication for secured Web sites. These techniques do not actually empower the user itself, or organisations handling personal data, with a fine-grained control on access of documents stored on mobile devices, e.g., copying, sharing, providing an anonymised view. More generally, they do not consider the current situation of the document gathered from the aggregation of several

contextual information, e.g., document accessed through an unusual place, without the presence of the usual user; document sent to an acquaintance that is not the friend of the original user; document is in a moving situation and not stationary; or document to be destroyed under the data protection law because no longer relevant for the organization.

In previous works, we defined the general notion of self-protecting context-aware content in the case of personal digital rights management [5], and provided the SmartContent approach [6] for pervasive and mobile environments, describing a generic model of self-protecting and context-aware active content that can act autonomously and protect itself against any unauthorized or unusual activity.

This paper discusses a refined model of SmartContent encapsulating content and context-aware encrypted policies. It proposes an architecture and actual implementation using the SAPERE context-aware middleware [7], a proof-of-concept using tablets and mobile phones showing the dynamic and context-aware access or restriction places on documents, a series of variants for usage involving remote servers or clouds. The current implementation protects content from accesses and manipulation actions requested from the application level. Future work will provide a complete protection, adding a supplementary layer at the operating system level as well. Access and manipulation requests from our proof-of-concept, currently involve situations gathered from contextual information such as neighbouring devices or location, and simple dynamic changes of policies. Future work will integrate more challenging contextual information such as social networks.

Section 2 discusses the current state of the art in self-protecting content, highlights their specific characteristics and vulnerability and strength against attacks. Section 3 presents the generic model and our approach. Section 4 shows the architecture and the implementation we provided. Section 5 shows our proof-of-concept deployed on mobile devices. Finally, Section 6 discusses variants on clouds or using remote servers, while Section 7 highlights current limitations and future works.

2. Related Works

2.1. Traditional Techniques

Techniques such as steganography or fingerprinting [8] (extracting key features of a document) allow to determine whether an image, text or video are actually copies of the original document but do not prevent their visualization or modification by unauthorized persons. Traditional digital rights management (DRM) techniques limit the viewing of protected content, but require special software or equipment, thereby limiting the flexibility. Most DRM systems found in media, enterprise and gaming rely on highly invasive and strict techniques, often closed or proprietary and hence significantly hamper user experience (flexibility, ease of use, etc.). In addition, most DRM systems are designed for large organizations, with little to no provision for people to use them for their own personal content protection.

Current DRM techniques manage sharable documents by resorting to distributed infrastructures composed of trusted servers. In the Cloud computing era such solutions are becoming more and more common to combine security of private data with the increasing need of making such information accessible to a particular range of users. This is the case, for example, of personal health records (PHRs), containing patient health profiles (diseases description, medications history, etc.) that are made accessible differently and simultaneously to doctors, pharmacists and relatives. The need to store PHRs and to identify and authorize users triggering specific actions on records requires the presence of a distributed infrastructure of servers and certification authorities. Key management issues can be challenging, as reported in [9], which presents a security model to securely share PHRs stored on semi-trusted servers by addressing such difficulties.

2.2. Context-Aware Solutions

Context-awareness paves the way for designing infrastructureless protect techniques, involving for instance a better flexibility in keys management processes by replacing the notion of private key by the notion of context.

For instance, in the scenario of personal health records, the context can identify the user who wants to access the internal data, avoiding a private key exchange process to characterize the users to whom the access is granted.

Nowadays context-awareness is getting more and more important especially in three security fields: (i) user authentication/identification (e.g., [10,11]), where context dimensions such as time and location, along with data generated by physical sensors, are considered passive source of data revealing additional information for straightening authentication/identification process; (ii) two-factor authentication on Web services, based on current ambient sound [12], where access to the Web site is granted, if in addition to login and password, the mobile phone of the user and its computer or laptop record the same ambient sound at the time of login; (iii) application usage control models (e.g., ConUCON [13]), which define fine-grained policies for applications execution and file access based on external (e.g., time, location) and internal (e.g., battery level) contextual information.

Nevertheless, current works on personal content protection do not properly consider context-awareness or they require expensive infrastructure not appropriate for deployment for personal usage because of the cost or the lack of flexibility.

Indeed, the self-protecting document (SPD) [14] uses a polarization key, which is highly bound to the user device, while the self-protecting container technology (DigiBox) [15] requires prior deployment of tamper-resistant hardware.

More recently, from the research field on cloud storage, context-aware solutions emerged. Munier et al. [16] designed and implemented self-protecting documents for cloud storage, where autonomic documents encapsulate data as well as additional information such as access and usage control, traceability and metadata information, or collaborative work management. This solution is specifically meant for cloud services and document sharing, it allows protecting groups of files. Access is granted on the basis of contextual information, available metadata and a license of use provided by the user.

Chen et al. [17] designed and implemented SelfProtected Object (SPO), which is an object built from original documents and access and action policies. Policies consider context such as access time and days, or IP address [18]. This solution requires the use of a remote trusted service to enforce the policies on the document upon access request. These two research works introduce a document oriented architecture for Enterprise DRM in which documents encapsulate both data and modular security mechanisms to access the internal content information. Security is achieved by executing an intermediary entity acting between the user and the internal information database. In both cases the intermediary unit is used to check user's actions against security policies through a validation process involving several internal purpose-specific security modules (e.g., access control module, trustworthiness management module, etc.).

In MULE (Mobile User Location-specific Encryption) [19] sensitive data is chippered by using a location-based key. The model aims at providing confidentiality in a complete transparent way for the user, who is not involved in any explicit interaction during the access to sensitive information. The authentication process is passive and performed automatically on the basis of the location of the device that must be equipped with pre-installed Trusted Location Devices, spare machines encapsulating micro-controllers that provide trusted-information about the current location. In MULE, location is the only existing contextual source, so it does not provide a more general architecture for context-aware confidentiality based on multiple sources of information.

SAINT [20] introduced a framework for Android devices behaving as an intermediary entity between applications and the policy manager of the operating system. The platform implements a context-aware decision module that limits the interactions between the system and a specific

application on the basis of a set of policies that the latter can specify. Such policies depicts some constraints that a generic vendor may want to enforce at runtime to implement safe execution conditions. Confidentiality is not addressed in such model.

Finally, Boukayoua et al. [21] defined a context-aware secure storage for Android devices that restricts data access according to predefined policies. Depending on the implementation, the storage container can be physically stored on the same device running the operating system or in an external tamperproof computational unit (e.g., smartcards). The interaction between an application and the secure storage is not transparent: it requires the instantiation of a secure channel compliant with a specific protocol that involves user's password issuing. Thus, context-awareness is employed as user-assistive security functionality and it does not represent the primary source of information for performing access decision mechanisms.

We observe that these different approaches do not jointly target content mobility, flexibility of policies and context-awareness, or they consider some of them secondary. Nevertheless, all such characteristics are needed in dynamic environments where content is on the move, within mobile devices or from clouds to local devices and vice-versa. Moreover, some of the approaches above require the execution of intermediary trusted security services on client-side, encouraging users to have confidence in the host system executing the code.

2.3. Analysis and Comparison

We report the main features of the frameworks cited above in Table 1. *Typology* defines the categories of models proposed by each framework, that can be of type *Access Control Model* (ACM), *Usage Control Model* (UCM), *Authentication Model* (AUM) or *Encryption Model* (ECM); *Context-aware solution* can be *Full* if the framework uses several contextual information sources to provide its functionalities, *Partial* if the sources are predefined or if they play a secondary role in the framework or *Absent* if the proposed solution is not context-aware; *Additional component units* is evaluated to *Yes* if the implementation of the framework needs intermediary software units to interact with users and applications, *No* otherwise; *Architecture* specifies the type of the framework architecture, that can be *Stand-alone*, *Cloud-oriented* or any combination of them in case of polyvalent solutions; *Collaborative document access* specifies if the framework is able to coordinate multiple simultaneous interactions on distributed copies of the same document; *Traceability/policy management* specifies if the framework exposes functionalities to trace the updates of a document and if it supports the concept of policy for expressing constraints w.r.t. access or usage of the secured content; *OS/device dependent* defines the dependency of the framework on a particular operating system or device with specific hardware capabilities.

Table 1. Comparison of the cited security frameworks.

Framework	Typology	Context-Aware Solution	Additional Component Units	Architecture	Collaborative Document Access	Traceability/Policy Management	OS/Device Dependent
SmartContent	**ACM, UCM, ECM**	**Full**	**Only for UCM**	**Stand-Alone**	**No**	**Policy Managment**	**No**
Sound-proof [12]	AUM	Partial	Yes	Cloud-oriented	/	/	Microphones
ConUCON [13]	ACM, UCM	Full	Yes	Stand-alone	No	Policy managment	Android based
SPD [14]	ECM	Partial	Yes	Stand-alone	No	None	No
SPO [17]	ACM, UCM	Partial	Yes	Stand-alone, Cloud-oriented	Yes	Both	No
Digibox [15]	ACM, UCM	Absent	Yes	Stand-alone	No	Both	No
MULE [19]	ECM	Partial	No	Stand-alone	No	None	Yes
SAINT [20]	ACM, UCM	Full	Yes	Stand-alone	No	Policy managment	Android based
Secure Storage [21]	ACM, UCM	Partial	Yes	Stand-alone	No	Policy managment	Android based

Table 2 reports our analysis of the main security attacks that can be launched against the above cited solutions. We do not mean to evaluate the security level or robustness of every single framework/model; we estimate instead the potential impact on the system in case a specific class of attacks is finally carried out. Also, instead of listing explicitly all possible kinds of risks and threats (e.g., [22]), we identified and classified attacks in four macro categories. *Context monitoring* collects all passive and close-in attacks that aim at monitoring the context to try to guess the values used in the authentication-encryption processes. Full context-aware solutions are usually prone to this type of threats, that can be carried out by sensing context sources and recording the evolution of contextual information (e.g., eavesdropping in case of contextual-data sent over a network). Context monitoring can drastically reduce the size of key spaces of encryption algorithms by several orders of magnitude (e.g., the private wifi network name of a victim can be easily sensed if the attacker is not too far from his home, even though the number of all possible network names is huge). *Context poisoning* includes all types of active attacks aiming at perturbing the contextual information sensed by real/virtual sensors. The typical goal is to induce a Denial of Service, for example by preventing the system from measuring a change in the access policies of a document. Mobile devices are also prone to energy starvation attacks, which may be induced by forcing continued variations in the contextual information (e.g., in case of policy-oriented framework performing a continuous monitoring of the environment). *Software components tampering* identifies all kinds of active attacks against framework software components aiming at modifying their internal logic. Beyond traditional attacks (e.g., buffer overflows), frameworks employing intermediary agents, which implement essential tasks (for example providing authentication or usage control), are prone to tampering if such components are not executed in sandbox or safe-containers. Similarly, the class of *Hardware components tampering* collects attacks against hardware components. Our terminology includes *critical* when a category of attacks can completely compromise data confidentiality and either data integrity or availability ([23]); *partially resilient* when data confidentiality is not affected and *resilient* if all such properties are preserved. More details about the security analysis of SmartContent are reported in Section 4.3.

Table 2. Comparison of attacks for the cited security frameworks.

Framework	Context Monitoring	Context Poisoning	Software Components Tampering	Hardware Components Tampering
SmartContent	**Critical**	**Critical**	**Critical for ACM,UCM Resilient for ECM**	**Critical**
Sound-proof [12]	Critical	Critical	Critical	Critical
ConUCON [13]	Critical	Critical	Critical	Critical
SPD [14]	Critical	Resilient	Resilient	Partially resilient
SPO [17]	Resilient	Partially resilient	Critical	Critical
Digibox [15]	/	/	Depending on the implementation	Resilient
MULE [19]	Critical	Resilient	Resilient	Resilient
SAINT [20]	Critical	Critical	Critical	Critical
Secure Storage [21]	Partially Resilient	Partially Resilient	Critical	Depending on the implementation

3. Generic Model and Approach

The approach we propose [5] considers a document as an active entity that decides on its own to reveal itself or not. The decision depends on the current environment in which the document is evolving and on policies set up by the document owner or controller. The document evaluates its context of use and denies access to its content if it determines that the situation and the intended action on the document is unusual or unauthorized according to the defined policy, as depicted in Figure 1. To implement such functionalities we define an access control model, usage control model and also an encryption model to provide confidentiality. A two levels protection shield enforces security: the innermost layer, intrinsically combined with the document to protect, assures confidentiality and integrity, empowering them with context information. The outermost layer resorts to a local trusted software component to implement access and usage control mechanisms. In some scenarios, the

trusted software component for the outermost level may be considered compromised, in this case our layered structure offers modular security functionalities, still giving the chance to select the most appropriate solution with respect to the available software infrastructure surrounding the document.

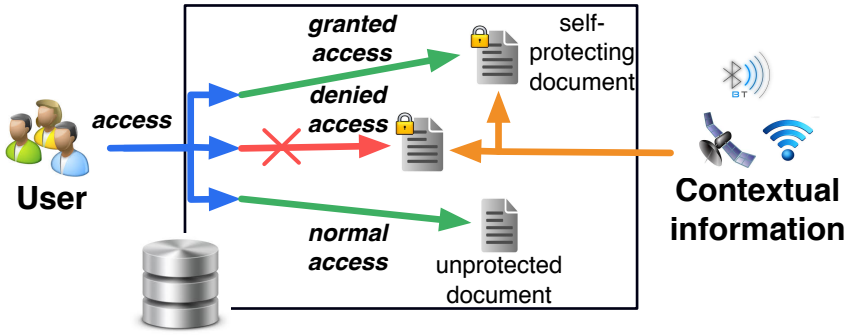


Figure 1. SmartContent approach.

A SmartContent document [6] is organized around the following entities, as shown on Figure 2: (i) the content itself, represented by a generic file (e.g., a picture, a spreadsheet, etc.); (ii) the sensed context (e.g., location, user biometric and behavioural profile, social network, camera pictures, noise, etc.), generated by physical sensors or third-party applications; (iii) user requests and manipulation actions (e.g., copy, delete, send, read, write, etc.), specifying the set of possible interactions between users and self-protecting documents; (iv) embedded or remotely acquired policies, associating user requests and actions with contextual values, specifying context conditions that must be satisfied to allow user interactions (e.g., GPS coordinates to allow access to the internal content); (v) reasoning unit, analysing the sensed context and policies to make decisions on which user interactions must be granted or denied.

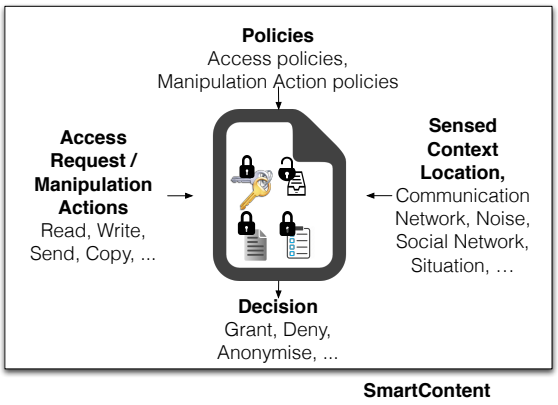


Figure 2. Generic SmartContent Model.

3.1. Contextual Information

SmartContent documents leverage two categories of contextual information.

- *Contextual-based information:* generated from distinct contextual data sources and based on physical or virtual sensor data such as biometric information, current location, current viewer, near Bluetooth/Wi-Fi devices, third-party applications data, etc. Each contextual data source is represented as a set of contextual data values C_{source}. The whole set of contextual data values forms a multidimensional space called *contextual space*, composed of as many dimensions as the total number of sources.

- *Situational-based information:* generated from the fusion of several contextual-based information gathered over some time periods. We call *situational space* the extension of the contextual space in which we consider also an additional temporal dimension. Generally speaking, a *state* of the system is a set of contextual data values belonging to distinct sources, including also a reference to a temporal dimension. A state is then represented by a point in the situational space. By sorting a set of states in ascending order with respect to the temporal dimension we obtain a sequence that represents an evolution of the system. We use the term *situation* to indicate a transformation of a set of states, i.e., information representative of the evolution of the system. $\mathcal{S}_{name}$ is the set of all values referring to the same typology of situation called *name*. Situational-values may require some manipulations to be representative; for example, when using GPS traces, single high-precision coordinates are difficult to be reproduced. In this case, situational-values belonging to particular ranges (there may be several dimensions) are replaced by reference values representative for such ranges (Figure 3).

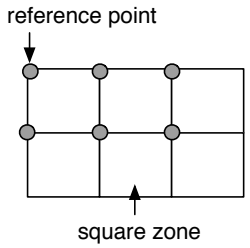


Figure 3. Reference system for GPS traces: all the points within a square zone are approximated with the coordinates of the reference point in the top left corner of the same area.

Example 1. *Sequences of GPS coordinates and an oscillation trajectory of a device (e.g., a smartphone) in the space are examples of situations: the former represents a discretization of the path between two places and involves no complex transformation of the sequence of states (i.e., the situation is itself a sequence of states); the latter is a situation that may be obtained by transforming a sequence of sensor data gathered by an accelerometer and a compass.*

3.2. Policies

Policies define criteria of interaction between users and SmartContent documents; they are rules that must be satisfied to manipulate SmartContent documents. More technically, policies can be considered expressions of predicates of type $name = value$ connected with logical connectives: *name* identifies a typology of situations or a contextual data source and $value$ is the associated description of the context that must be satisfied. We can divide policies into two categories.

- *Reading policies:* define the rules to access to the protected content of a SmartContent document. They identify subspaces of the situational and the contextual space that must be detected to read the internal content. Reading policies form the basic level of protection offered by SmartContent documents and they are intrinsically implemented within the documents.
- *Manipulation policies:* define the rules to restrict manipulation actions on SmartContent documents, like local or remote copies, renaming, etc. Manipulation policies are optional and resort to a reasoning unit trusted by the owner of the document.

Policies have names to be identified. Reading policies represent the main security functionality of SmartContent documents: they resort to encryption schemas and they do not need software components to be interpreted, i.e., they are intrinsically implemented in the way the internal content is stored. Manipulation policies, instead, represent an extra security layer and need additional software

components to be provided. An example list of reading and manipulation policies is reported in Table 3.

Table 3. List of policies.

Reading Policies
readable-when: the content is readable (decrypted) when the conditions expressed by the predicates are satisfied.
Manipulation policies
readable-until: the content is readable until the conditions expressed by the predicates are satisfied. **writable-until**: the content is writable until the conditions expressed by the predicates are satisfied. **allowed-remote-copies**: the set of nodes to which the document can be sent. **allowed-local-copies**: the set of local folders where the file can be copied.

Example 2. *A SmartContent document stored in a smartphone is created to be accessed and shared only among the employees of an office at the first floor of a building. In the rooms of the office there are two available Wi-Fi networks,* `netA` *and* `netB` *and the difference in altitude between the ground floor and the first floor is 5 m. To access the content, the document must sense a specific sequence of movements through the GPS sensor: such a sequence represents the trajectory of an employee that accesses the main door of the building through the principal walkaway. The document should also be accessible only in the time slot 8:30–19:00. Such description of the environment can be used to define the following policies.*

readable-when {
wifi-nets = {`netA`,`netB`}
and
gps-sequence = $(46.1763879, 6.1399586) \cdot (46.1763881, 6.1399554) \cdot (46.176387, 6.139956)$
and
wifi-sig-strength = –60;–50
and
time-slot = 8:30;19:00
and
altitude-variation = 5}

readable-until{
wifi-nets = {`netA`,`netB`}
and
wifi-sig-strength = –60;–50
and
time-slot = *8:30;19:00*
and
altitude-variation = 0}

allowed-remote-copies{
nodes = {employee1,...,employeeN}}

The block within the statement **readable-when** *identifies a reading policy: it states that the internal content can be read if and only if: (1) two wi-fi networks are surrounding the environment (contextual-based information); (2) their signal strength is in the range* $-60; -50$ *dBm (contextual-based information); (3) the*

right sequence of movements is detected (situational-based information); (4) the time slot corresponds to the desired one (contextual-based information) and (5) the smartphone has detected a 5 meters variation in the altitude (situational-based information). We consider the case of a generic employee entering the building through a door at the ground floor. The block with the statement **readable-until** *identifies a manipulation policy: the internal content is readable as long as the smartphone keeps detecting the same networks and no variations in the altitudes are perceived. Finally, the block* **allowed-remote-copies** *defines the devices of the employees to whom the SmartContent document can be sent (manipulation policy).*

The differences between reading policies and manipulation ones are reflected in the structure of a SmartContent document.

3.3. Internal Structure, Creation and Access to SmartContent Documents

A SmartContent document is composed of four parts: the encrypted content, the encrypted manipulation policies, the predicates of the reading policies and a set of encrypted keys used to decrypt the internal content. The reading policies are used to encrypt the protected content: the contextual and situational values within the predicates are combined (according to the logical connectives) to create a key for a symmetric encryption algorithm. Such values are then discarded after the encryption and the predicates are stored in the document. The predicates permit identifying the sources of context to use during the decryption process: if the current contextual and situational values match the ones used during the creation of the document, their combination yields to the regeneration of the key and the decryption can take place.

The internal structure reflects the components used during the creation process of the document, as shown on Figure 4 and explained below.

I The creator of the SmartContent document selects an unprotected document and specifies its policies by defining (a) a list of predicates; (b) some logical connectives and (c) some values $v_1, ..., v_k$ belonging to the contextual data sources $\mathcal{C}_{source-1}, ..., \mathcal{C}_{source-n}$ and to the situations $\mathcal{S}_{name-1}, ..., \mathcal{S}_{name-m}$.

II A randomly generated symmetric key k_c is used to encrypt the unprotected document and the manipulation policies with a symmetric encryption algorithm (e.g., *AES*).

III Depending on the logical connectives, values $v_1, ..., v_k$ are combined (for example by using a hash function like *SHA-3*) to create one or more encrypted versions of k_c. These keys are used to recover k_c depending on the contextual and situational values available at a given time. Managing logical connectives efficiently and secret sharing techniques can reduce the number of keys. The contextual and situational spaces represent the key space of the symmetric encryption schema, so $\mathcal{C}_{source-1}, ..., \mathcal{C}_{source-n}, \mathcal{S}_{name-1}, ..., \mathcal{S}_{name-m}$ should be sets sufficiently large to assure a robust key space.

IV All the encrypted objects are stored in a new file along with the reading policy predicates devoid of contextual and situational information values. According to the structure described so far, reading policy predicates are stored in the document without being encrypted: this fact does not affect the security of the document because the contextual and situational values used to create the encryption keys are unknown. Manipulation policy predicates, instead, are always encrypted within the document because they carry on also the associated values.

The process for accessing a SmartContent document is as follows:

I By analysing the unencrypted predicates of reading policies, a set of contextual data sources $\mathcal{C}_{source-1}, ..., \mathcal{C}_{source-n}$ and situations $\mathcal{S}_{name-1}, ..., \mathcal{S}_{name-m}$ are selected to obtain the set of values $v'_1, ..., v'_k$. Such a set identifies the current sensed context, i.e., the subspace of the contextual-situational space describing the environment in which the system is currently operative.

II If $v'_1, ..., v'_k$ satisfies the readable policies then at least one of the encrypted versions of $\|_c$ can be decrypted. In this case the environment satisfies the criteria imposed by the policies.

III With $\|_c$, the internal content of the document and the manipulation policies can be finally decrypted.

At this point a local trusted software component (reasoning unit) can be charged to monitor the context and the commands issued by users, comparing them against the manipulation policies. For situational-based information, the SmartContent Manager must keep track of the evolution of situational values. This process can be performed by analysing the predicates of policies associated with situations $\mathcal{S}_{name-1}, ..., \mathcal{S}_{name-m}$, in order to understand how situational-based information must be managed by the application.

Depending on the operating system, a trusted software component can deny the access to a self-containing document by resorting to the file system functionalities concerning file permissions and access rights. For example, being executed as *root* it can become the owner of the file and limit the access to other users. In this case, the trusted software component becomes an intermediary unit between the user and the SmartContent document, acting as a filter for users' commands (e.g., by blocking commands for local copies).

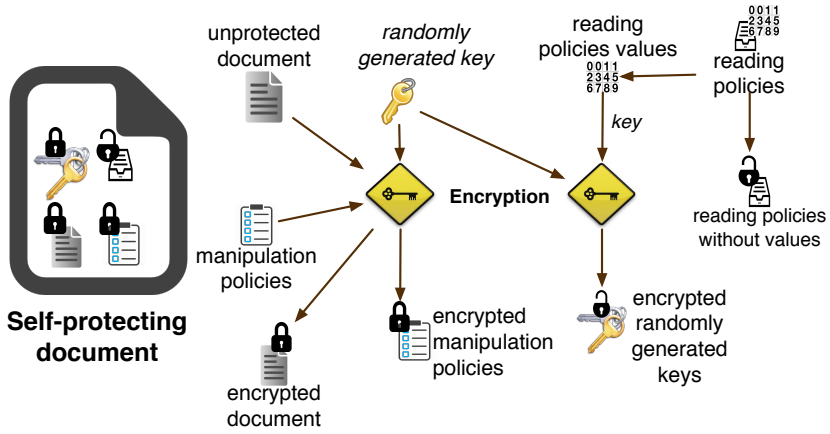


Figure 4. Structure and creation of a SmartContent document.

4. Architecture and Implementation

We have implemented a proof-of-concept of SmartContent documents specifically for mobile devices; in the following section we analyse its architecture and we describe further security improvements to make it usable in additional scenarios.

4.1. SAPERE Middleware

As detailed in Section 4.2, our solution uses the SAPERE middleware [24], a coordination framework offering functionalities for data and context representation, aggregation and exchange to design distributed context- and situation-aware multi-agent systems. SAPERE has been used to implement and deploy self-organizing systems for pervasive scenarios [25], providing useful nature-inspired mechanisms [26] to design and develop distributed applications [27].

The framework is based on the following internal components: (1) *Live semantics annotations (LSAs)*, active tuples encapsulating information about entities of the pervasive ecosystem. They are considered active entities because they can interact among each other under the execution of chemical inspired laws named *Eco-laws*.

LSAs can represent context information provided by context sources or any information provided by agents. LSAs are dynamically aggregate and processed by eco-laws; (2) *LSA Space*, tuple space

running on each device of the pervasive system serving as a container for hosting LSAs and for sharing LSAs among agents; (3) *Agents*, which represent the interfaces with the *LSA Space* of external components taking part to the coordination process (e.g., sensors, services, applications, etc.). Every agent is associated with an LSA and implements the external component logic for managing the notifications triggered during the execution of the chemical laws; (4) *Eco-laws*: digital substratum providing a set of bio-inspired mechanisms that regulate the manipulation of stored information. The bio-inspired mechanisms resemble chemical and biological laws proper of natural systems; in particular, the middleware provides four bio-inspired processes: (i) instantiation of relationships among LSAs (*bonding*); (ii) aggregation of LSAs according to predefined mathematical operators (*aggregate*); (iii) reduction of information relevance (*decay*); (iv) diffusion of LSAs towards remote LSA Spaces (*diffusion*). Eco-laws are fired by specifying special operators embodied in the contents of LSAs. For example, to request a bond coordination rules, agents inserts either "?" or " $*$ " as property value of a tuple. Every time a bond is established or removed and every time a property value involved in a bond is updated an event of activation is delivered to the agent exhibiting the value "?" (" $*$ "), which handles the event executing code external to the tuple space. This is the main way coordination is performed among the agents. For example, a bond will be created between $T_1 = (city = ?, weather = !)$ and $T_2 = (city = "NewYork")$ because of property *city*. In this case, the agent implementing an hypothetical service described by the tuple T_1 will be invoked, generating the weather forecast for "New York". This will result in the generation of a new tuple, e.g., $T_3 = (city = "NewYork", weather = "sunny")$.

4.2. Architecture

The architecture of our application is composed of five components (Figure 5).

- *SAPERE middleware*: used to have a common model for representing, collecting and spreading contextual and situational information. Every contextual data source (e.g., GPS sensor, accelerometer sensor, third-party application such as calendar, address book, etc.) is managed by a *contextual agent*, which injects the associated contextual information in the tuple space under the form of LSA. Resorting to the functionalities provided by the Eco-laws, specialized agents named *situation profilers* produce situational information by aggregating LSAs injected by contextual agents. LSA can also encapsulate SmartContent documents shared with other nodes of the network, handled by specialized agents named *file managers*. This feature assures that information shared among nodes is managed by resorting to the functionalities provided by middleware. SAPERE is used to handle contextual and situation information and to send and receive SmartContent documents towards and from other nodes of the network. An agent called *Context/Situation (CS) Manager Agent*, which connects the framework to the *Context/Situation manager*, represents the interface with the remaining components of the architecture.
- *Context/Situation manager*: component charged to manage contextual and situational information, acting as an intermediary unit between the *SAPERE middleware*, the *SmartContent Controller* and the *Policy manager*. It also notifies the *SmartContent Controller* when a document is received from another node, and analogously it is invoked when a document must be sent through the network.
- *Policy manager*: component charged to manage the reading and manipulation policies of the SmartContent documents existing in the system. It is connected to the *Context/Situation manager* to access the information about the context and to the *Controller* to inform it about changes in the manipulation policies. The Policy manager is charged to manage both the policies of an existing SmartContent document as well as the new policies introduced by a user during the creation of a new document. When the policies of a document become unsatisfied, the component notifies the *controller*, which requests the *Encryption manager* to re-encrypt the document.

- *SmartContent Controller*: main unit orchestrating all other components of the application. Through an interface it also receives the external commands issued by users (e.g., creation of a new SmartContent document).
- *Encryption manager*: component managing the authorized manipulation actions on a SmartContent document. Given that it implements the encryption/decryption algorithms and the routines to access the internal content and policies, it acts as the interface between the file system and the *SmartContent Controller*, implementing the security functionalities to encrypt, decrypt, copy, delete, rename a SmartContent document when a notification is generated by the *SmartContent Controller*. Moreover, it is also charged to recover the (internal) Manipulation Policies of the document once accessible, passing them to the *SmartContent Controller*.

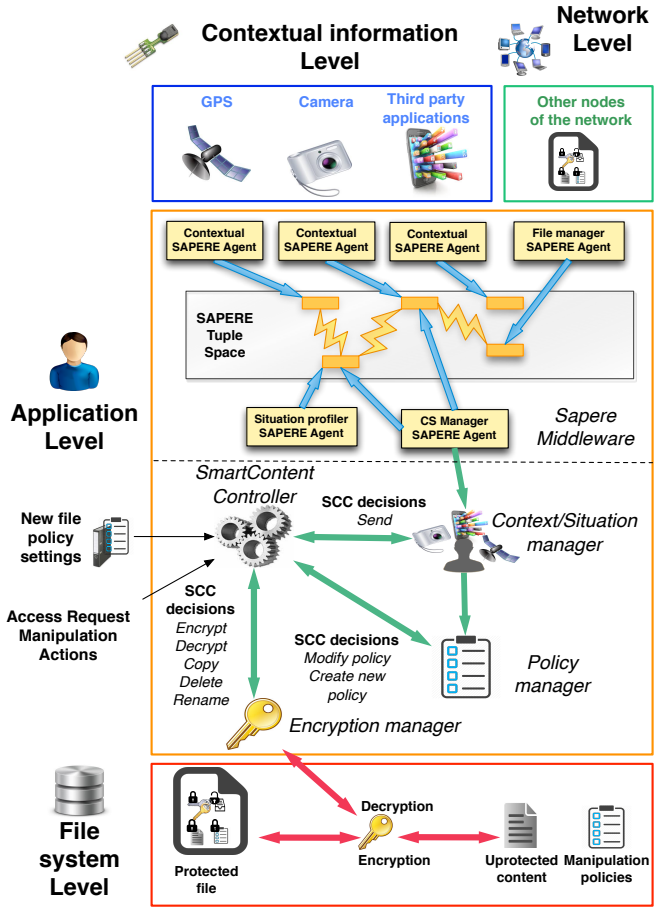


Figure 5. Architecture of an application managing SmartContent documents.

4.3. Security Analysis

Like other full context-aware solutions, context monitoring attacks can be critical for SmartContent. This vulnerability can be tackled by selecting multiple contextual sources that are more difficult to be sensed, such as biometric ones. Due to the mobility of self-protecting documents we do not require the presence of Trusted Platform Modules and we assume that the implementation of the architecture analysed in Section 4 can be executed on untrusted devices such as traditional smartphones and tablets, making SmartContent also prone to context poisoning and hardware components tampering attacks. In case the software components are compromised, the only available security functionality is represented by reading policies. This means that the access and control usage model of SmartContent are prone to software components tampering, whereas the encryption model is resilient. Some kinds

of these attacks can be tackled by protecting the application at the file system level, for example preventing it from being replaced with malformed versions.

4.4. Implementation

The SAPERE middleware is distributed under the form of an open-source Java platform, composed of a kernel implementing the main functionalities of the coordination model and a set of architecture-dependent libraries, providing support for several operating systems; in particular, we used the extension of the middleware for Android devices.

5. Proof-of-Concept and Demonstration

Based on the architecture presented in the previous point, we implemented a demo application in the context of mobile devices: the application is executed on a Samsung Tab 2 tablet (main tablet) and is able to convert pictures in SmartContent documents; the goal is to prove that they can dynamically reveal or conceal themselves based on sensed context. A second tablet of the same type provides a Bluetooth interface exploited in the policies of the examples presented below; similarly, a second instance of the SAPERE Middleware is executed on a Samsung S4 smartphone to convey network messages to the main tablet through the Wi-Fi network. Table 4 reports the list of SAPERE developed for the demonstrations. Two Contextual SAPERE agents are used on the main tablet to collect the contextual information associated with the Bluetooth and Wi-Fi networks. Such information is then managed by the Contextual Information Manager Agent, the interface between the SAPERE Tuple Space and the Context / Situation manager. A fourth SAPERE agent is deployed on the smartphone to send network messages to the main tablet.

Table 4. List of SAPERE agents used in the demonstration.

Type	Device	Description
Contextual SAPERE Agent	Main tablet	Bluetooth manager used to discover bluetooth devices.
Contextual SAPERE Agent	Main tablet	Network manager used to receive messages sent over a Wi-Fi network.
CI Manager SAPERE Agent	Main tablet	Manages the contextual information generated by the contextual agents.
SAPERE Agent	Smartphone	Generates and sends messages over the Wi-Fi network.

A video of the three tests analysed in this section is available online.
In the first one (Figure 7) we use the following policies:

readable-when {
`bluetooth-neighs=` {tablet2}}
readable-until{
`bluetooth-neighs=` {tablet2}}

where `bluetooth-neighs= {tablet2}` is satisfied if and only if a Bluetooth device with name `tablet2` is detected by the main tablet. The reading policy is equal to the manipulation one. In Figure 6 we show the transitions for the main tablet system. When the user selects a picture to convert in self-protected document there are no Bluetooth devices available; after an interaction between the Context/Situation Manager and the Policy Manager, the SmartContent Controller invokes the

Encryption Manager to protect the document by concealing the picture (Figure 7a). When the second tablet with Bluetooth name `tablet2` is placed near the main one, the Contextual SAPERE Agent detects its presence and injects the name of the device in the Tuple Space. The Context/Situation manager passes such information to the Policy Manager, that interacts with the SmartContent Controller. A decryption key is generated on the basis of the Bluetooth device name; such key is correct because the device name matches the one in the access policy, so the decryption process succeeds and picture becomes visible (Figure 7b). Finally, when the second device turns off the Bluetooth interface, the Contextual SAPERE Agent removes the tuple previously created and a removal notification is triggered by the Tuple Space. The notification is propagated up to the Policy Manager, that reacts informing the SmartContent Controller to protect again the picture because its manipulation policy is unsatisfied.

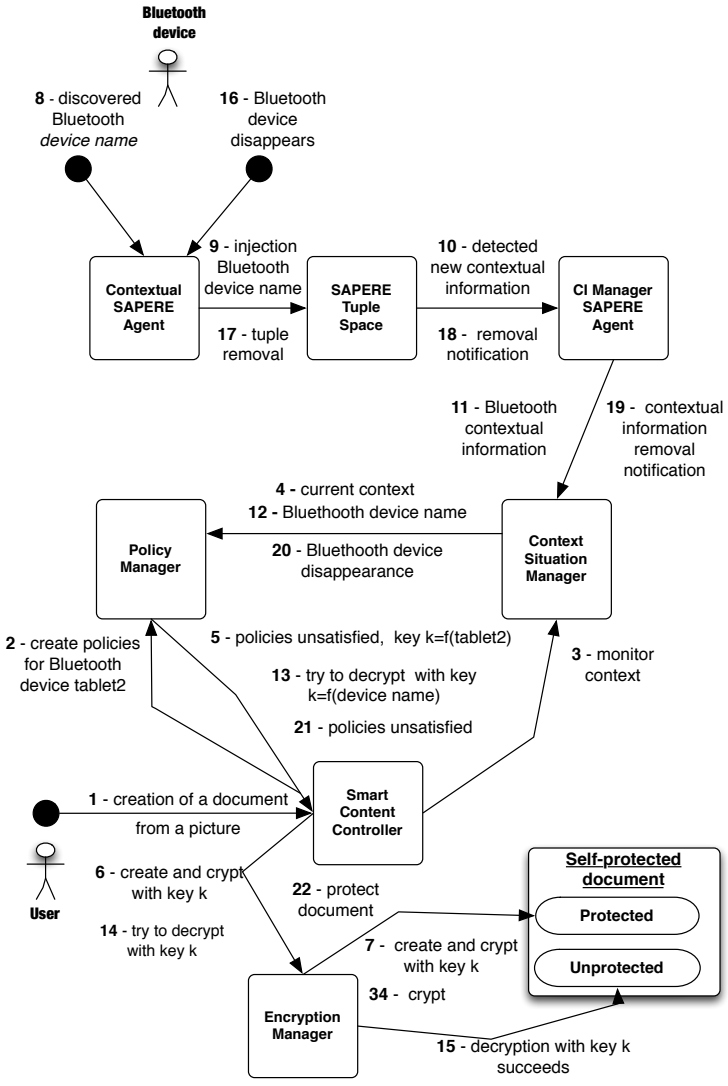
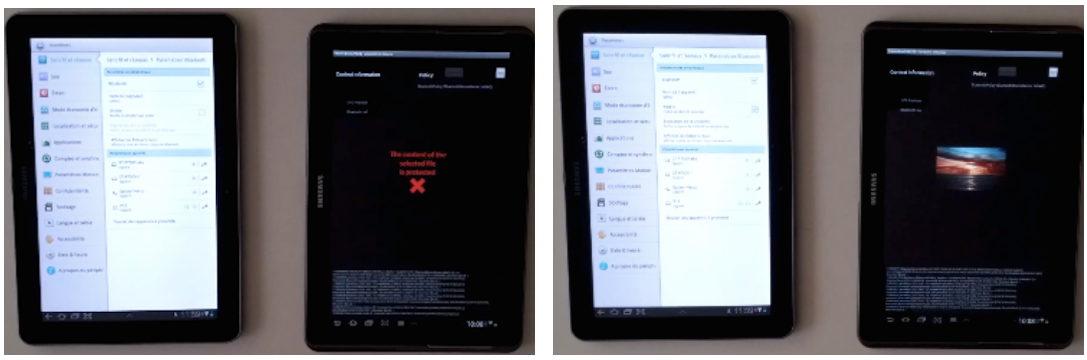


Figure 6. Transitions for the system in the first demonstration.



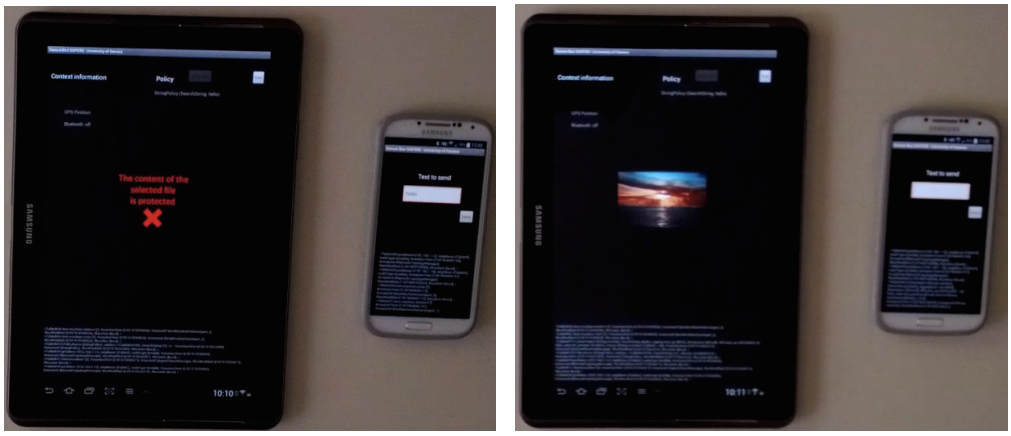
(**a**) (**b**)

Figure 7. Policy based on Bluetooth device presence. (**a**) Access denied; (**b**) Access granted.

In the second test (Figure 8) we use the following policies:

readable-when {
`network-msg = 'hello'`}
readable-until{
`network-msg = 'hello'`
and
`lifetime = '5'`}

In this case the contextual information is represented by a message (with content `'hello'`) sent by a smartphone through a SAPERE agent. Interactions similar to the ones of Figure 6 take place also in this second test, so they are omitted. The policy `network-msg` is satisfied if and only if such message is received in the main tablet. The message is a readable string but it could be any other information conveyed through a secure channel. Also in this case, when the SmartContent document is created the message is not received yet and the picture is concealed (Figure 8a); as soon as the message is spread by the smartphone, a notification is propagated from the Contextual SAPERE agent up to the Encryption Controller, that decrypting the content successfully makes the picture visible (Figure 8b). The message has a predefined lifetime of a few seconds (`lifetime` property): when it is deleted from the main tablet a removal notification triggers the execution of Policy Manager, that decides to protect and hide the picture again because its manipulation property is not longer satisfied.



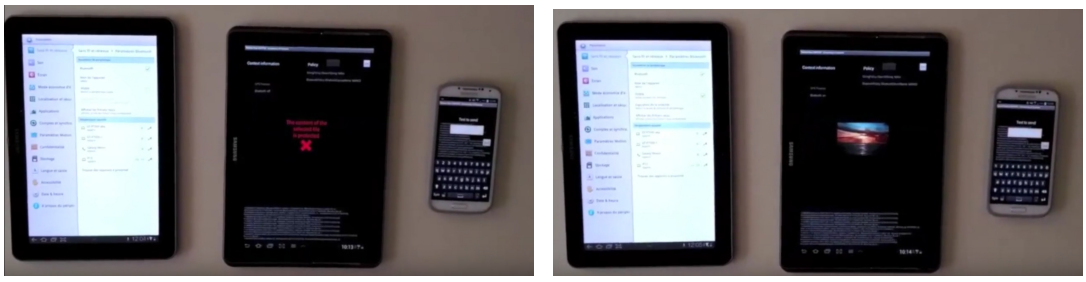
(**a**) (**b**)

Figure 8. Policy based on network messages. (**a**) Access denied; (**b**) Access granted.

In the last test we combine the policies of the previous cases, obtaining the following rules.

readable-when {
`bluetooth-neighs` = {`tablet2`}
and
`network-msg` = `'hello'`}
readable-until{
`bluetooth-neighs` = {`tablet2`}
and
`network-msg` = `'hello'`}

Given the presence of the logical connective *and*, the image is revealed only if the main tablet detects the existence of a device with Bluetooth name `tablet2` and if, at the same time, it receives a network message with content `'hello'` (Figure 9). Also in this case the reading policies and the manipulation ones are identical. In Table 5 and Figure 10 we report the average encryption times of AES 128 and AES 256 when used to encrypt files with several predefined sizes (being a symmetric algorithm, we assume that decryption times have the same order of magnitude). The cryptographic suites have been tested on the real devices used during the demonstrations and on an Android emulator armeabi-v7a executed on a MacBook Pro Quad core i7 2.4 Ghz with 8GB RAM DDR3 1.6 Ghz. As we can see, on recent smartphones the average time to protect a document of 4MB is approximately half a second. This means that, assuming the trend showed in Figure 10, a whole folder containing 1 GB of data can be secured in about 3 minutes. Even though such waiting time may be considered excessive, it represents the current state of the art for huge dimension files (the reader can refer to [21] for a comparison of performances of the most used encryption libraries).



(**a**) (**b**)

Figure 9. Policy based on Bluetooth device presence and network messages. (**a**) Access denied; (**b**) Access granted.

Table 5. Average encryption time for several file sizes (milliseconds).

	4 KB	**40 KB**	**400 KB**	**4 MB**
Emulator AES 128	18.92	74.54	294.77	1631.59
Emulator AES 256	20.4	79.56	309.15	1952.23
Samsung Galaxy Tab S2 AES 128	13.95	54.4	217.3	761.51
Samsung Galaxy Tab S2 AES 256	14.98	58.4	227.94	975.89
Samsung Galaxy S4 AES 128	4.75	18.45	75	458.8
Samsung Galaxy S4 AES 256	5.4	21.26	87.6	590.78

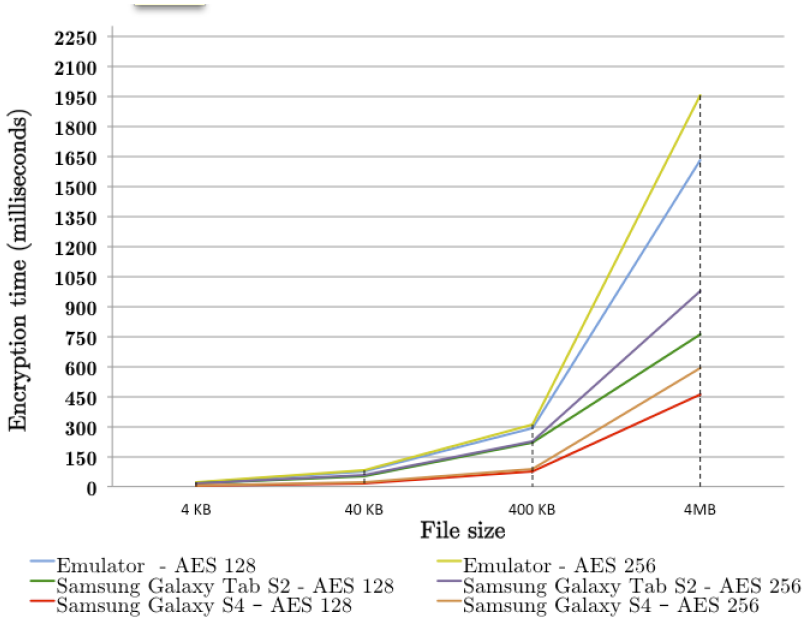


Figure 10. Average encryption time for several file sizes.

6. Variants on Cloud and Remote Servers

In order to accommodate various usages required by private or professional needs, such as documents on mobile devices, but also documents accessed through cloud infrastructures, we will consider appropriate variants of the above described model and corresponding implementations for different cases, such as: (1) documents carried on mobile devices with local sensed context and inferred situations, policies and controller embedded within the device (Figure 11); (2) documents carried on mobile devices, local context sent to remote server, and any global context (e.g., multiple requests from different locations) useful for the document, policies and actual decisions to grant access stored in and provided by a remote server (Figure 12); (3) documents and policies stored on clouds or remote servers, viewing access occurring through a mobile device, local context sent to the remote server, any useful global context stored at the remote server (Figure 13). Any combination of the above can also be considered. The use of the SAPERE middleware serves both locally within the mobile device for computing local context as well as on the remote server or cloud for computing global context information. Examples discussed in this paper consider single files, the approach can protect folders or groups of files in a similar manner.

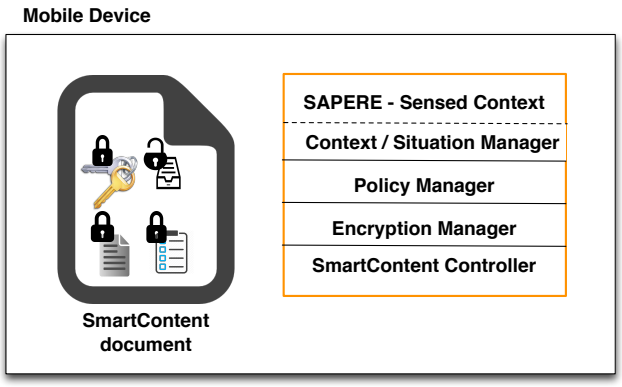


Figure 11. Documents carried and protected on mobile devices only.

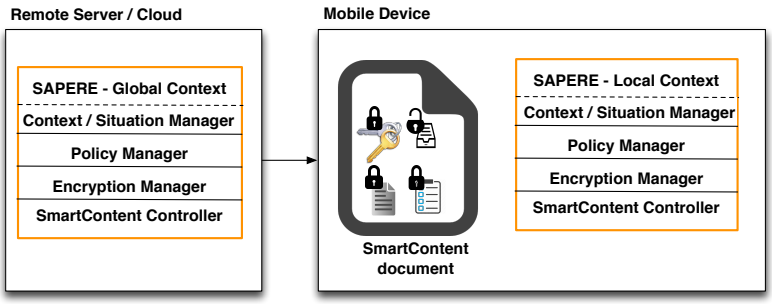


Figure 12. Policies and decisions granted from both local and remote locations.

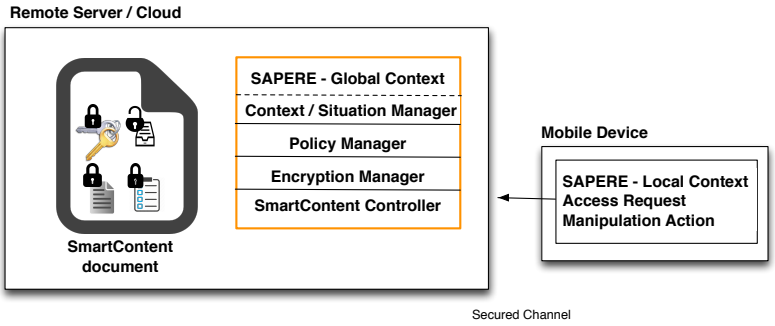


Figure 13. Documents stored and protected remotely.

7. Further Security Improvements

As described in Section 3.3, a trusted agent (in our implementation the SmartContent Controller) should resort to the functionalities provided by the file system to implement some manipulation policies. If the trusted agent is a traditional application with administrator privileges it can easily deny the access to the document to other software components. The correct implementation of the manipulation policies should also take into account the problem of revoking the access to an unprotected document as soon as the manipulation rules becomes unsatisfied; in fact, there may be programs still reading the internal content even though the file system permissions changed. To solve such problem in Linux-based operating systems like Android, the trusted agent could resort to existing virtual file system functionalities to close the file handles of the processes still reading the protected file. Nevertheless, such functionalities may not be provided by the kernel version used by a particular retailer.

Another solution consists in implementing the closure of file descriptors within a custom external kernel module (Figure 14). When the module receives a notification of the trusted agent specifying the file name to close, it accesses internal kernel structures and closes the file descriptors related to the protected file. Given that external kernel modules can be loaded at run-time, this solution is less invasive because it does not require to recompile the kernel (which is usually optimized by the retailer for a given class of devices). In this way, the trusted agent and the module can be delivered under the form of a unique application.

In order to accommodate flexibility of policies description and usage, we plan to capture richer context coming from physical (e.g., captors on mobile phone), logical (e.g., agenda of user) or virtual sensors (e.g., social network links) available through mobile or stationary devices on which the document is stored. Additionally, an efficient reasoning engine will be developed for identifying actual advanced situations (e.g., document is on a stolen device) from current context information gathered from physical or virtual sensors.

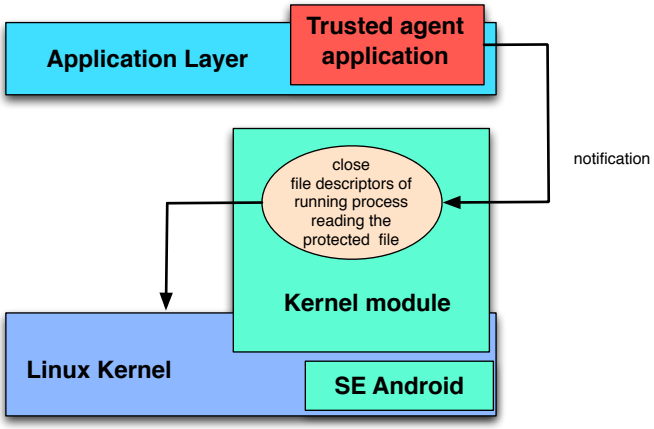


Figure 14. Kernel module used to close existing handles to protected files.

8. Conclusions

We propose SmartContent, a self-protecting content approach, to ensure privacy that is context- and situation-aware, policy-based and ensures compliance to laws and organizations norms. The main characteristics and strengths of the SmartContent approach are: (1) documents are fully autonomous when stored on mobile devices, there is no need for a trusted remote service granting access; (2) context- and situation-aware access manipulation action decisions; (3) approach primarily tailored for mobile devices, extendable to clouds and remote servers; (4) dynamically changing policies; (5) policies and required context- situation- embedded in the encryption keys; (6) open-source implementation using a context-aware middleware.

Preliminary work with context information such as presence or absence of wireless connection, or specific context information dynamically provided, demonstrates the usefulness of the concept. Besides the current application level, we intend to fully develop the idea at the operating system level to make it actually usable in both private and professional settings. This work also applies to data held by organizations having to comply with data protection laws, copyright or the right to forget. It corresponds to the concept of "Privacy embedded in the design recommending a proactive protection of privacy, user-centered, and integrated in the design of the system that protects the data.

The scope of self-protecting documents ranges from individuals for their private usage, to professionals responsible of data inside organizations, to professionals handling data as part of their work (e.g., bank employees, policemen, health professionals, public administration employees, HR departments employees, etc.). We can also identify the following cases of usages: (1) avoiding errors while handling files; (2) data owner empowerment and control; (3) accessing data in case of emergency; (4) protecting from malicious attacks.

Acknowledgments: Part of this work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873. Preliminary work has been undertaken jointly with Akla-Esso Tchao.

Author Contributions: Francesco Luca De Angelis contributed to the architecture of SmartContent documents and to the implementation of case studies; Giovanna Di Marzo contributed to the definition of the SmartContent model and all its variants.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Johnson, N.F.; Jajodia, S. Exploring Steganography: Seeing the Unseen. *Computer* **1998**, *31*, 26–34.
2. Anderson, R.J.; Petitcolas, F.A. On the Limits of Steganography. *IEEE J. Sel. A Commun.* **2006**, *16*, 474–481.

3. Irwin, J. Digital Rights Management: The Open Mobile Alliance DRM Specifications. *Inf. Secur. Tech. Rep.* **2004**, *9*, 22–31.
4. Liu, Q.; Safavi-Naini, R.; Sheppard, N.P. Digital Rights Management for Content Distribution. In *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003*; Australian Computer Society, Inc.: Darlinghurst, Australia, 2003; Volume 21, pp. 49–58.
5. Tchao, A.; Di Marzo, G.; Morin, J.H. Personal DRM (PDRM)—A Self-Protecting Content Approach. In *Digital Rights Management: Technology, Standards and Applications*; Hartung, F., Kalker, T., Shiguo, L., Eds.; CRC Press: London, UK, 2012.
6. Tchao, A.E.; Di Marzo Serugendo, G. SmartContent: A self-protecting and context-aware active content. In Proceedings of the 2nd Workshop on Challenges for Achieving Self-Awareness in Autonomic Systems (AWARE) at Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO12), Lyon, France, 10–14 September 2012; pp.151–156.
7. Zambonelli, F.; Castelli, G.; Ferrari, L.; Mamei, M.; Rosi, A.; Di Marzo Serugendo, G.; Risoldi, M.; Tchao, A.E.; Dobson, S.; Stevenson, G.; et al. Self-Aware Pervasive Service Ecosystems. *Procedia CS* **2011**, *7*, 197–199.
8. Broder, A. On the Resemblance and Containment of Documents. In Proceedings of the International Conference on Compression and Complexity of Sequences 1997, Positano, Salerno, Italy, 11–13 June 1997; pp. 21–29.
9. Li, M.; Yu, S.; Zheng, Y.; Ren, K.; Lou, W. Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 131–143.
10. Hayashi, E.; Das, S.; Amini, S.; Hong, J.; Oakley, I. CASA: Context-aware Scalable Authentication. In Proceedings of the Ninth Symposium on Usable Privacy and Security, Newcastle, UK, 24–26 July 2013; pp. 3:1–3:10.
11. Goel, D.; Kher, E.; Joag, S.; Mujumdar, V.; Griss, M.; Dey, A.K. Context-Aware Authentication Framework. In *Mobile Computing, Applications, and Services*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 35, pp. 26–41.
12. Karapanos, N.; Marforio, C.; Soriente, C.; Capkun, S. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound. In Proceedings of the 24rd USENIX Security Symposium (USENIX Security 15), USENIX Association, Washington, DC, USA, 12–14 August 2015; pp. 483–498.
13. Bai, G.; Gu, L.; Feng, T.; Guo, Y.; Chen, X. Context-Aware Usage Control for Android. In Proceedings of the 6th Iternational Conference on Security and Privacy in Communication Networks (SecureComm 2010), Singapore, 7–9 September 2010; pp.326–343.
14. Ram, P.; Ta, T.; Wang, X. Self-Protecting Documents. EP Patent 0999488, 2005.
15. Sibert, O.; Bernstein, D.; Van Wie, D. Digibox: A self-protecting container for information commerce. In Proceedings of the First USENIX Workshop on Electronic Commerce, New York, NY, USA, 11–12 July 1995; pp. 1–13.
16. Munier, M.; Lalanne, V.; Ricarde, M. Self-Protecting Documents for Cloud Storage Security. In Proceedings of the TrustCom–11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Liverpool, UK, 25–27 June 2012; pp. 1231–1238.
17. Chen, S.; Thilakanathan, D.; Xu, D.; Nepal, S.; Calvo, R. Self Protecting Data Sharing Using Generic Policies. In Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Shenzhen, Guangdong, China, 4–7 May 2015; pp. 1197–1200.
18. Thilakanathan, D.; Chen, S.; Nepal, S.; Calvo, R. SafeProtect: Controlled Data Sharing with User-Defined Policies in Cloud-based Collaborative Environment. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 301–315.
19. Studer, A.; Perrig, A. Mobile User Location-specific Encryption (MULE): Using Your Office As Your Password. In Proceedings of the Third ACM Conference on Wireless Network Security, Hoboken, NJ, USA, 22–24 March 2010; pp. 151–162.
20. May, M.J.; Bhargavan, K. Towards Unified Authorization for Android. In *Engineering Secure Softwware and Systems*; Lecture Notes in Computer Science Series 7781; Springer: Berlin/Heidelberg, Germany, 2013.
21. Boukayoua, F.; Lapon, J.; Decker, B.D.; Naessens, V. Secure Storage on Android with Context-Aware Access Control. In Proceedings of the 15th IFIP TC 6/TC 11 International Conference Communications and Multimedia Security, CMS 2014, Aveiro, Portugal, 25–26 September 2014; pp. 46–59.
22. M, S.; G, P. Mobile Device Security: A Survey on Mobile Device Threats, Vulnerabilities and their Defensive Mechanism. *Int. J. Comput. Appl.* **2012**, *56*, 24–29.

23. Padmavathi, G.; Shanmugapriya, D. A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks. *Int. J. Comput. Sci. Inf. Secur.* **2009**, *4*, ISSN 1947 5500.
24. Castelli, G.; Mamei, M.; Rosi, A.; Zambonelli, F. Engineering Pervasive Service Ecosystems: The SAPERE Approach. *TAAS* **2015**, *10*, 1:1–1:27.
25. Zambonelli, F.; Omicini, A.; Anzengruber, B.; Castelli, G.; DeAngelis, F.L.; di Marzo Serugendo, G.; Dobson, S.; Fernandez-Marquez, J.L.; Ferscha, A.; Mamei, M.; et al. Developing Pervasive Multi-Agent Systems with Nature-Inspired Coordination. *Pervasive Mob. Comput.* **2015**, *17*, 236–252.
26. Fernández, J.L.; Serugendo, G.D.M.; Montagna, S.; Viroli, M.; Arcos, J.L. Description and composition of bio-inspired design patterns: A complete overview. *Nat. Comput.* **2013**, *12*, 43–67.
27. Castelli, G.; Mamei, M.; Rosi, A.; Zambonelli, F. How to Develop Pervasive Social Applications with the SAPERE Middleware. *Comput. Inform.* **2015**, *34*, 185–209.