# Enabling Secure Ad-hoc Communication using Context-Aware Security Services

**— Extended Abstract —**

Narendar Shankar[*]
University of Maryland
narendar@cs.umd.edu

Dirk Balfanz
Palo Alto Research Center
balfanz@parc.com

## 1. Introduction

It is a stated goal of the ubiquitous computing research thrust to make the computer "disappear". One of the most intrusive aspects about computers today is security management. Thinking about security places an immense burden on users (*e.g.,* Is it safe to open this attachment? Should I accept this cookie? Should I let Internet Explorer remember my password?, *etc.*). Moreover, managing security settings is cumbersome, difficult to understand, and often considered a hassle that is in the way of getting work done [8].

Making security management "disappear" does not only accomplish a goal of ubiquitous computing – we believe that it can, in fact, also make things more secure. If users are bothered less often with dialog boxes that they don't understand, then they are less likely to make bad decisions. Likewise, if users don't have to go through a difficult security setup, then they are less likely to skip that cumbersome step.

In this paper, we propose a step toward making security management disappear in certain situations.

Imagine the following situation: you are at a meeting in a conference room and would like to share a sensitive document you just received with members from various organizations in the conference room. Traditionally, you would have to have some sort of *a priori* trust information from the intended recipients of your sensitive message (such as their public keys). Exchanging this trust information is a cumbersome step, and not everybody in the room may participate in the same Public Key Infrastructure (PKI).

Recently, mechanisms have been suggested to exchange trust information on-the-fly in an *ad-hoc* manner [2], which would reduce the need for cumbersome setup steps and also eliminate the need for an all-embracing PKI.

But the ultimate automation of this process would be if, simply by entering the conference room, all participants of the meeting became members of some sort of secure group communication scheme they could use to communicate with each other. No setup would be necessary, no trust information would have to be exchanged explicitly. Instead, you would simply send, say, an email to `room123@mycorp.com`, which would be encrypted and sent to all people in the room. By virtue of being in room 123 you would automatically posses the keys needed to encrypt the email, and the other participants would have the keys to decrypt it. Other people not participating in the meeting would not be able to read your email.

[*]This work was done while working at the Palo Alto Research Center.

In this paper, we propose and describe a system that allows you to do just that, and - more generally - to use contextual information to enable secure ad-hoc communication. In other words, we track the context of various devices and group devices based on context. After the devices have been grouped based on context, a security service uses the group information to form secure associations.

Also, one would like to maintain secure associations across different contexts. Continuing with the same example, the participants of the conference would like to continue their secure group communication from the conference room to a lunch party. This automatically puts them in a different context. Instead of establishing new secure associations, it would be desirable to continue with the same secure association in a different context. This requires tracking of various contexts and maintaining secure associations over the contexts.

The main contributions of this paper are

1. A new framework for expressing, using, and combining contextual information easily and efficiently.
2. A generic *contextual security service* for securing ad-hoc communication using contextual information obtained from the above framework.
3. A way to extend the traditional meaning of context to enable secure associations over what would be traditionally considered multiple contexts.

The rest of this paper is structured as follows: In Section 2 we explain the notion of context, context views, context view maps, and how secure associations are established using contextual information. In Section 3 we present an overview of the architecture and some interesting applications. We then wrap up with a comparison of related work in Section 4 and conclusions in Section 5.

## 2. Preliminaries

In this section, we first talk about context views and provide the intuition for using contextual information for making security decisions.

### 2.1. Moving from Context to Context Views

In this paper, though we talk about multiple *contexts*, we actually refer to *context views*. An entity's context comprises an infinite amount of information (location, temperature, noise level, orientation in space, battery level, *etc.*) that changes over time. On one hand, this makes it hard to define the term "context" (see [3] for a discussion of various attempts to define context). On the other hand, any given application will only be interested in a miniscule fraction of that entity's context (say, just the entity's location and its temperature). We call this limited information about an entity a *context view*. Intuitively, a context view is a small part of an entity's context, typically the part that a particular application is interested in. A context view is both dependent on the application observing the context and the time when the context is observed.

We choose to leave the term "context" undefined, and instead define *context view maps* as the fundamental concept in our system.

**Definition 1** *Let $\mathcal{T}$ be the (ordered) set of time,* i.e., *$t \in \mathcal{T}$ is a point in time. Let $E$ and $C$ be sets. We call a function $m : \mathcal{T} \times E \longrightarrow C$ a* context view map. *We call $E$ the set of* entities *and $C$ the* context view type. *We call the elements of $C$* context views. *We call the set $E \times C$ the* map type *of $m$.*

EXAMPLES. If an application is interested in the location of entities, then the context view type $C$ could be the set $Geo := Long \times Lat$ of geographical coordinates (where $Long$ is the set of all possible longitudes and $Lat$ is the set of all possible latitudes). If an application is interested in the location and temperature of an entity, the context view type could be $Geo \times Temp$ (where $Temp$ is the set of all possible temperatures). If an application is interested in which room in a building a given entity is in, then the context view type $C$ could be the set of all rooms in that building.

The set of entities could be the set of all known users of a system, or all possible IP addresses, all possible network adapter hardware addresses, *etc.*

A context view map might, for example, map from IP addresses into geographical coordinates (in which case $C = Geo$), or room numbers in a building, providing information about the location of devices.

Each entity can exist in multiple context views. For example, one application might be interested in the location of a particular device, while another application might be interested in which devices that device is talking to. While both pieces of information are part of the device's context, the two applications would view that context through their respective views, one mapping from IP addresses to geographical coordinates, and the other mapping from IP addresses to sets of IP addresses.

Formally speaking, the two application may employ different context view maps to represent an entity's context:

$$\begin{aligned} m_1 &: \mathcal{T} \times IP \longrightarrow Geo \\ m_2 &: \mathcal{T} \times IP \longrightarrow \mathfrak{P}(IP) \end{aligned}$$

(where $\mathfrak{P}(X)$ is the power set of $X$.)

The same device $d$ would, at time $t_0$, be mapped to a location $m_1(t_0,d)$ by one application, and to a set of communication peers $m_2(t_0,d)$ by another application.

### 2.2. Building Services using Context Views

Context view maps are used as building blocks for the entire architecture. An application uses multiple maps that it is *interested* in and fits them into a *subscription hierarchy*. Once the context views have been fit into the hierarchy, the application can use a single resultant context view map or multiple resultant context view maps, depending on the nature of the application.

Formally, we define the composition of context view maps as follows:

**Definition 2** *Let $m_1 : \mathcal{T} \times E_1 \longrightarrow E_2$ and $m_2 : \mathcal{T} \times E_2 \longrightarrow C$ be two context view maps. Let $e_1 \in E_1$ and $t \in \mathcal{T}$. We define the composition of context view maps $m_1 \circ m_2 : \mathcal{T} \times E_1 \longrightarrow C$ as follows:*

$$m_1 \circ m_2(t,e_1) = m_2(t, m_1(t,e_1))$$

EXAMPLE. If we have a context view map $m_1 : \mathcal{T} \times IP \longrightarrow Geo$ that maps from (IP addresses of) devices to their geographical locations, and a map $m_2 : \mathcal{T} \times Geo \longrightarrow R$ that maps from geographical locations to the room numbers in a building that those locations correspond to,[1] then it is easy to provide a context view map that maps from IP addresses to room numbers by simply composing the two maps: $m_1 \circ m_2 : \mathcal{T} \times IP \longrightarrow R$.

### 2.3. Where does security fit in?

Using context views, security can be easily abstracted out into a service. To describe how we provide security in our contextual framework, a few preliminary definitions are necessary:

[1]Presumably, this mapping would not actually depend on the time parameter.

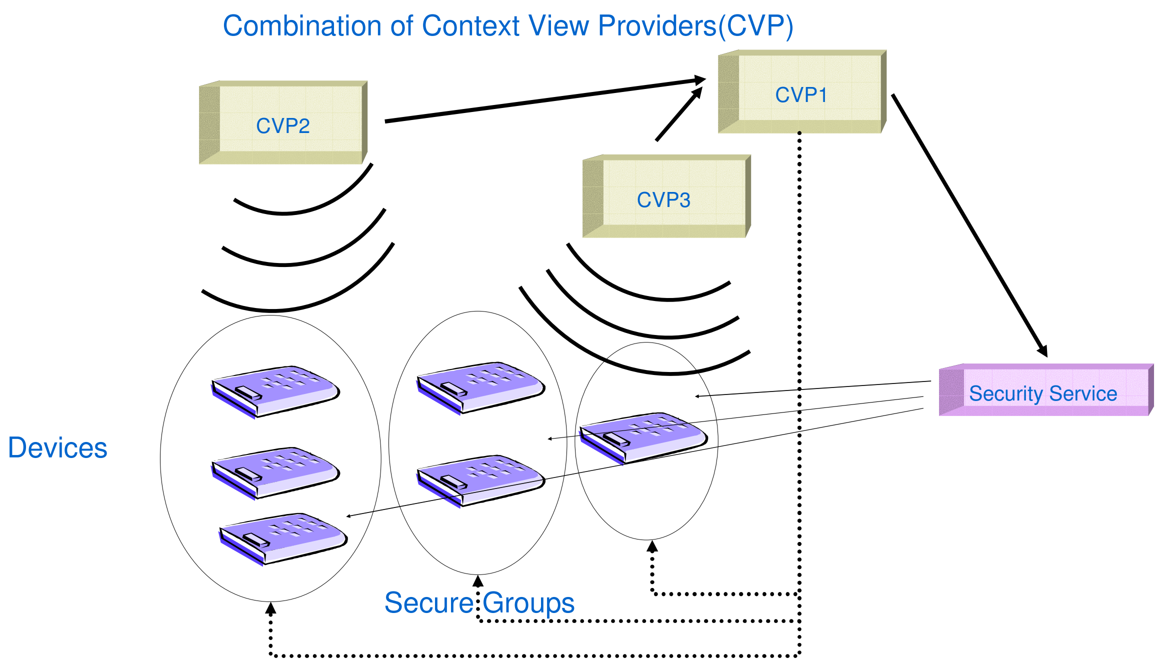

**Figure 1. Security Service Architecture. The context view providers CVP2 and CVP3 observe devices and their contexts. CVP1 subscribes to the context views and publishes its own context view map. That map induces context view associations among the devices (dotted arrows). The security service subscribes to CVP1, learns the context view associations, and manages the security for each association (thin arrows).**

**Definition 3** *Let $m : E \longrightarrow C$ be a context view map. For any given context view $c \in C$, we define the* context view association *of $c$ under $m$* (written as $|c|_m$) *as the set of all entities that are in the same context view $c$. This set will vary with time. Therefore, we have $|c|_m : \mathcal{T} \longrightarrow \mathfrak{P}(E)$, and for any point in time $t$*

$$|c|_m(t) = \{e|\ m(t,e) = c\}$$

EXAMPLES. If $m : IP \longrightarrow R$ maps from devices to room numbers of the rooms the devices are in, then $|r|_m(t)$ is the set of all devices that are in room $r$ at time $t$. More generally, $|r|_m$ represents the devices in room $r$ and is a function that maps points in time to sets of IP addresses.

**Definition 4** *Let $m : E \longrightarrow C$ be a context view map from $E$ to $C$.*

*We denote the set of all context view associations as $E/m$:*

$$E/m = \{|c|_m \mid c \in C\}$$

*We can view $E/m$ as a function from time to sets of entities. For each point in time $t$ we have*

$$E/m(t) = \{|c|_m(t) \mid c \in C\}$$

EXAMPLES. For our $m$ above, $IP/m(t)$ would be the set of all context view associations under $m$ at time $t$, *i.e.*, one element of it would the set of IP addresses of the devices in room 123 at time $t$, another element would be the set of IP addresses of the devices in room 124 at time $t$, and so forth. There is an obvious one-to-one mapping between rooms and the set of devices that are in those rooms.

**Definition 5** *A* contextual security service for $m$ *is a service that manages secure group communication for each context view association in $E/m$ (recall that each element in $E/m$ is the set of entities that share identical context view values). In particular, a contextual security service manages the context view association for each possible context view value as a secure group.*

EXAMPLES. A contextual security service for the $m$ mentioned in our previous examples would be a service that manages secure group communication for all devices in room 123, separately from that for all devices in room 124, etc.

Note that a contextual security service has to deal with dynamic group memberships – as entities change their contexts (*e.g.*, move around in a building), the context view associations lose or gain members (remember that $E/m$ is a function over time). A contextual security service therefore uses well-established group key distribution protocols to distribute short-lived keys to all members of a context view association.

The group key distribution protocol has to be done over a secure channel. To establish this secure channel, we could utilize a pre-existing PKI. However, this is not necessary in all circumstances. A different way of doing this would be to exchange all trust information needed for setting up the secure channel (between the security service and the members of various context view associations) at some earlier point. In our example, this could correspond to exchanging public keys between the security service and the various devices at the reception desk (when a person/device enters the building) over a location limited channel [2].

Once the contextual security service distributes keys to the current members of the various context view associations, they can use these keys to securely communicate with each other.

See Figure 1 for a summary of this process. Those entities that fall into the same resultant context view can be treated as members belonging to the same group and the security service then helps negotiate security parameters between the group members.

As mentioned before, using contextual information enables secure ad-hoc communication between group members. The context view associations become *secure associations*.

### 2.4. Extending Secure Associations across Contexts

Sometimes it may be desirable to extend secure associations across contexts. For example, entities that formed a secure association while co-located in a certain conference room in a building might wish to continue that secure association even after they leave that room.

Because of our very general definition of context views, this notion can actually be captured by a single context view map. We simply define a new "artificial" context view type, say the binary set $\{0, 1\}$. We map all entities that were once co-located in the conference room to the context view value "1", and all other entities to the context view value "0". Now, the generic security service described above, under this map, will exhibit the desired functionality, and associate entities even after they leave the conference room.

We now look at the architecture, which represents a practical implementation of the theoretical concepts introduced above.

## 3. Design and Applications

### 3.1. Design

In this section, we describe the design of the security service architecture. This can easily be extrapolated to other services. As shown in Figure 1, there are multiple devices. There is a set of *context view providers* (CVPs), arranged in a subscription hierarchy. The CVPs are implementations of the context view maps introduced in Section 2. A CVP publishes an XML encoding of the map type it implements. Other CVPs can subscribe to the information provided by one or more CVPs, essentially performing a composition of context view maps.

The security service uses one context view provider and maps the various devices into a set of bins, which correspond to context view associations. Once devices have been arranged in their specific bins, short-lived group keys can be given to these groups.

Also, as entities keep migrating from their base context, we have to keep track of the various contexts. Returning to our example, for a secure association with a base context being a particular conference room, the set of valid contexts may include all locations inside the building. In other words, if a secure group has been formed inside the conference room, it can be valid so long as all members are inside the premises of the building. To capture this we introduce the concept of an *artificial context view provider* (ACVP). An ACVP uses other CVPs to create an "artificial context". In our example, the ACVP that does the above will use two CVPs, one which maps a device to the rooms they are in, and another which maps a device to the one-bit information whether it is inside the building or not. It will in turn export a context view map that implements the desired behavior.

### 3.2. Sample Applications

Some of the sample applications we envision are:

1. Instant secure email, which uses the above security service architecture to create instant secure mailing lists. For example, an email sent to *room123@mycorp.com* should reach all members currently in that conference room.

2. Secure file sharing applications, which allow members in context view associations to securely share important files or documents.

3. Instant secure groupware, like web conferencing tools and so on.

### 3.3. Context Authentication

We would like to point out that our work is not concerned with *authentication* of contexts, *i.e.*, we assume that the CVPs report correct and trustworthy context views. Obviously, if the CVPs *don't* provide correct and trustworthy context views, the secure associations managed by the contextual security service may be compromised.

We see the authentication of contexts as an orthogonal issue, but would like to point out two things:

- There has been some progress in context authentication. See, for example [5].

- Even if the CVPs are unreliable, or untrustworthy, with some user interaction false information provided by a CVP can detected. The users can simply ask the security service to identify all other users that it placed in the same association, and then verify that this coincides with their expectations.

## 4. Related Work

Our work addresses the problem of enabling secure communication among a group of entities in an ad-hoc manner, which is both user friendly and scalable.

There has been a lot of prior work on toolkits for capturing context, with the most popular one being Dey's Context Toolkit [7, 1]. Our architecture for capturing contextual information is similar to Dey's work but for the definitions of context, context views and the kinds of mappings we have defined. There have also been lots of other similar initiatives [4, 10, 6], which do address particular needs but our design has been motivated by a *simple framework* for grouping devices based on context.

As far as securing ad-hoc communication goes, the resurrecting duckling model by Stajano and Anderson [9] introduces a preliminary idea for securing ad-hoc communication. The work by Balfanz *et al.* [2] builds on this idea and proposes a solution using location-limited channels. Both ideas are innovative and help solve the problem of securing ad-hoc communication. In this paper, we address a similar issue for securing ad-hoc communication, but for larger groups. Other solutions don't deal with that problem efficiently. For example, in [2], trust information has to be exchanged over a location limited channel like IR. This can take quite some time to establish a secure group because every member of a group has to exchange information with every other member in the group.

We believe that the solution proposed in this paper is a reasonable way to secure ad-hoc group communication and can be used in an effective manner in many situations.

## 5. Conclusions

In this paper we presented a new scheme for securing ad-hoc communication using contextual information. We explained how to use contextual information to form secure groups. Unlike previous work, our solution easily scales to large groups. The security management all but disappears and is handled automatically by a generic security service. Below we summarize the novel aspects of our work:

1. The use of a context-view architecture to create subscription hierarchies.

2. The use of context views to enable a generic contextual security service to create and maintain secure ad-hoc associations.

3. Identifying interesting applications, which use the secure ad-hoc associations.

We believe that our solution is a stepping stone for future work on using contextual information for building secure infrastructures.

## References

[1] G. D. Abowd, A. K. Dey, R. Orr, and J. A. Brotherton. Context-awareness in wearable and ubiquitous computing. In *ISWC*, pages 179–180, 1997.

[2] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of Network and Distributed System Security Symposium 2002 (NDSS'02)*, San Diego, CA, February 2002.

[3] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, December 2000.

[4] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: Web presence for the real world.

[5] T. Kindberg, K. Zhang, and N. Shankar. Context authentication using constrained channels. In *Proceedings of IEEE WMCSA 2002*, New York, June 2002.

[6] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science*, 2201:56–68, 2001.

[7] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[8] D. K. Smetters and R. E. Grinter. Moving from the design of usable security technologies to the design of useful secure applications. In *New Security Paradigms Workshop '02*. ACM, 2002.

[9] F. Stajano and R. J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th Security Protocols Workshop*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–194, Cambridge, United Kingdom, 1999. Springer-Verlag, Berlin Germany.

[10] P. Tandler. Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In *Proceedings of UbiComp 2001: Ubiquitous Computing*, number 2201 in LNCS, pages 96–115. Springer Verlag, Heidelberg, 2001.