Toward Trustworthy Ubiquitous Computing Environments

Weidong Cui, Yitao Duan, and Kai Wei

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
{wdc, duan, kwei}@EECS.Berkeley.EDU

***Abstract*—In a Ubiquitous Computing environment, sensors are actively collecting data, much of which can be very sensitive. The environment will often be unfamiliar to the users. They will not have a trust relationship with the owners of the environment as they might with their local system administrator appropriate for handling their private information. Data will often be streaming at high rates (video and audio) and it must be dealt with in real-time. Effective solutions for controlling access to data in ubicomp settings remain to be developed. Dealing with these issues will be a central challenge for ubicomp for some time to come. But in the mean time we propose some simple design principles which address several of these issues. We illustrate them through the design of a smart room capture system we are building at Berkeley. The key principle is "data discretion", which grants access to information only to individuals who would have "real-world" access to the data. We propose a hierarchical key based cryptosystem to enforce the "data discretion" principle. Unlike traditional ACL based access control systems, our scheme essentially embeds access rights of legitimate users in the data. Our approach has the advantages of (1) strong data security based on cryptography; (2) simple and flexible sharing among legitimate users; and (3) easy verification of stated policies. We have implemented a prototype system in the Berkeley Smart Room testbed and demonstrated the efficiency of our scheme. Finally we describe ongoing work toward a trustworthy ubicomp environment whose discretion is realistically checkable.**

I. Introduction

Ubiquitous Computing (ubicomp) has been hailed as the new paradigm for next generation computing. A ubicomp environment is typically envisioned as a space populated with large number of invisible, collaborating computers, sensors and actuators interacting with user-worn gadgets. Data about individuals who are in the environment is constantly being generated, transmitted and stored. Much of the data can be quite sensitive. Protecting these private data is a central concern for the users to have a trust relationship with the environment. There are a few challenges that make data security in Ubicomp settings different from other system protection:

1) The environment is often unfamiliar to the users. They will not have a trust relationship with the owners of the environment as they might with their local system administrator appropriate for handling their private information.
2) Data are often generated dynamically and streaming at high rates (video and audio) and must be processed in real-time.
3) Users' access rights change dynamically with respect to their relationship with the mechanisms by which data are generated. For example, a number of users can form an ad hoc group and record their meeting using a camera that is administrated by the environment. They should only have accesses to the video produced during the meeting period but not others. The system must be able to associate a piece of information with the correct set of users while it is being produced.
4) Typical usage of user's data involves sharing among a group of people [1]. Any protection scheme implemented must allow efficient sharing among legitimate users.

Unless there is a clearly stated protection policy and scheme in place, together with provisions for convincing verification, users will not be comfortable trusting the infrastructure with their data. And this will hinder the acceptance and limit the usefulness of such systems.

For those data that are generated by the user (e.g., documents produced on users' laptops), protection is easy. Clients can perform some protocol to agree upon one session key and encrypt files by themselves and upload the encrypted files to the server. They can also store the session keys on the server using a variant of the scheme that will be described in Section IV. Since the server never sees the session key or the clear data, it has no way to leak them.

In this paper we are concerned with protection of user data that are generated by the infrastructure. This case is tricky because the users have to trust the system to some degree (e.g., the fidelity of the sensor data) since they rely on the system to perform certain operations (e.g., data generation, transmission, and encryption). We do not want to encourage "blind trust" from the user. Rather, we would like to design the system in such a way that it promotes user trust and confidence. We consider a typical ubicomp environment, in this case a smart room augmented with a variety of sensors. The issues we address in this paper are:

1) Protection of the user data generated and maintained by the environment.
2) Privacy of individuals who use the environment.
3) Ability of legitimate users to make use of data recorded in the environment.
4) Dealing with high-speed streams of data.
5) Trustworthiness of the environment and users who have access to the data it captures.

We propose an approach to protect user data in a dynamic and ad hoc environment. Our scheme makes use of both secret key and public key cryptography and can achieve efficiency high enough to be implemented using today's commodity hardware and software, and deal with steams of audio and video data. It is by no means a complete solution to data security in ubicomp environments, rather it offers a simpler and more efficient alternative to ACLs and represents a first step toward making a ubicomp environment trustworthy.

The rest of the paper is organized as follows. In Section II we describe our testbed smart room. Section III presents related work which motivates our privacy principle. In Section IV we present the data discretion principle and describe the protection scheme that enforce this principle and how it is applied to our smart room. Section V gives performance evaluation of our prototype system. Section VI presents extensions and possible techniques for improving and validating privacy compliance in ubicomp environments. Finally Section VII summarizes and describes directions for future research.

II. Smart Room Testbed

We have begun building a smart room equipped with sensing and able to record user activity. The room is a small meeting room with a conventional whiteboard, a Smart Technologies Smartboard™ with projector, four video cameras, and a Philips I-CODE™ RFID tag reader [2]. There is a single entrance to the room, and the tag reader antenna is mounted around this entrance so that it can record entry and exit by any user carrying an I-CODE tag (it also provides security and can record any piece of tagged equipment that might be removed from the room). The tags provide reliable recognition of users entering the room, although users without tags will not be noticed by the tag reader. It can of course be fooled by tags which move through the doorway when not attached to their users. The cameras capture images from the room continuously, and send them to a data server. Another computer drives the Smart-board, and this computer runs a logging program that records all significant user activity using the board. The smart room is representative of typical ubicomp environments and serves as a good testbed for our privacy principles. The images and log data represent dynamically generated user data that we seek to protect. We will provide a simple playback application that explores the privacy protections. Section V will give more details about the prototype system and the experiments we conducted on the smart room testbed.

III. Related Work

The Platform for Privacy Preferences, or P3P for short, is developed by the World Wide Web Consortium (W3C) and emerging as an industry standard for web servers to specify and communicate their privacy policies to a browser in an automated way [3]. P3P is intended to give users more control over the use of personal information on Web sites they visit, by allowing Web sites to describe their data collection practice in a machine readable way and enabling users or client software to make decisions on accepting or rejecting the policies. While P3P simplifies, to some extent, the process of judging the acceptability of a site's practice, users will lose control over their personal data once the information is on the server side.

Other research projects seek to protect user anonymity in communication and data capturing [4] [5] They basically follow the rules and practice established in Internet Privacy [3] [6], which focus on obscuring user's IP address, and extend them into ubiquitous computing context. The Mist Project at UIUC strives to anonymize users' communication in ubicomp environments [5]. It utilizes a hierarchy of "Mist Routers" that perform "handle-based routing" to preserve privacy and hide information about the source and destination. Users will expose their identities to part of the system, but their locations will be concealed. The EuroPARC's RAVE [7] presents a networked node model where interactions are defined by connections established between nodes. It emphasizes two principles in preserving privacy: control and feedback. The former empowers users to stipulate what information they project and who can access it while the latter allows the users to be informed of the capturing and usage of their information. Privacy preserving is achieved by controlling connection capabilities. While this is a reasonable model for protecting transient information, it is not clear how permanent data is protected. Also it represents an "untrustworthy" environment for visitors who have no knowledge of how it works.

In [8], Marc Langheinrich laid out some principles for design of privacy-respecting ubicomp systems. Any realistic design requires a combination of technical solutions, legal and policy constraints, and practical detection and enforcement schemes. In our work, we strive to build technical solutions that can be combined with realistic inspection and enforcement practice to lead to privacy-safe systems. Langheinrich stresses the importance of including privacy considerations in the early stage of system design process. He notices that current approaches to privacy issues in ubicomp environment are mostly ad hoc and application specific. To remedy this, he develops six principles for guiding privacy-aware ubiquitous system design. Langheinrich's framework largely follows the existing and emerging legislations and stipulates the desired system features and user's social behavior. The six principles are:

1) Notice: users should always be aware of what data is being collected.
2) Choice and Consent: and should be able to choose whether it is used.
3) Anonymity, Pseudonymity: should apply when identity is not needed.
4) Meeting Expectations: systems should mimic real-world norms.
5) Security: different amounts of protection depending on the situation.
6) Access and Recourse: users should have access to data about them.

Langheinrich's principles are a good starting point for our work. We next describe a design approach that supports several of the principles directly, and makes it easy to build systems which support all of them. Our approach differs from other work in that instead of building complex access control into a system (which must not be compromised), we focus simultaneously on *protecting* the sensitive information, but *exposing* the protected data stream in ways that allow inspection and monitoring of privacy compliance.

IV. SYSTEM DESIGN: THE DATA DISCRETION PRINCIPLE

If we take Langheinrich's principles 4 and 6 together, meeting expectations and access and recourse, we derive a principle we call data discretion:

Data Discretion: Users should always have access to, and control of (recorded or live) information that would be available to them in "real-world" situations. They should not have direct access in other situations.

So for instance, users should have access to information recorded in a smart room that while they were in the room. They should not have access to information recorded in that room while they were not present. They should also have control of this information, and be able to use it however they please.

By making access to the user a requirement, we can make it easier to satisfy the other privacy principles 1, 2, 3 and 5. That is because we can require all accesses to user data to be routed to the user, and therefore we involve the user in all decisions about use of their data. In our current smart room, the only application is review of video or queries about who was in the room when the user was there. Although support for individual access only seems restrictive, in other work [1] [9] [10] we have demonstrated techniques that allow users to collaboratively combine their data without losing privacy. Put together, these methods provide the means to maximally protect private information, but still use the information for useful collaborative tasks.

There is some subtlety in making the discretion principle work. Particularly in a smart room setting, we would like to keep the identity of the people who were in the room a secret from those who were not in the room at that time. This means that users who were not in the room at the time should not have access to information recorded in the room at that time, nor should they be able to find out who does have access, because that is equivalent to knowing who was in the room. This rules out many access control methods, which expose the set of users who have access to files to system administrators or others who can penetrate the operating system. We prefer methods based on cryptography which make it impossible to gain access to sensitive information even with control of the machine and operating system.

A. Basics

Our method is based on hybrid secret-key and public-key cryptography. We assume that all data is stored in files that represent short time intervals. For instance, each video file contains a short sequences of images of one second or so, or logs from the Smartboard in files that span about a second, etc. Every user file is encrypted with a unique, randomly generated secret key. This secret key is then encrypted by the public keys of the file owners. We will describe the scheme in details in Section IV-C.

A user joins our system by registering his or her public key and choosing a recognition tag that the system can detect once it is within the vicinity of a sensor. The recognition tag is used by the system to detect user's presence so that it can retrieve related information (e.g., user's public key). We define recognition tag in an abstract sense in that it does not have to be a physical tag and can be implemented with any technology (e.g., RFID tag or biometric). The user is assumed to wear the recognition tag.

Note this approach is different from access control lists (ACL) that are typically used to protect user data whose owner is predeterminable. Our system only needs to know the association between user data and its access key(s) and can function perfectly well without knowing identification of the holder of a particular tag. No attempt is made to connect a tag with a user's real-world identity, or even his or her system-wide electronic identity. User anonymity is therefore protected.

B. Establishing Information Ownership

The first step in protecting user data is to ascertain the natural owners of the information. In a dynamic environment such as ubiquitous computing context, this has to be done while the information is being generated. We assume that the system has the ability to detect a user's recognition tag when they are present in the environment. Other identification technologies (e.g., face and fingerprint recognition) are expanding rapidly, and there already exist several that are mature enough to offer non-intrusive detection at reasonable cost. However, some of these technologies rely on recognition of information that are unique to a user (e.g., fingerprint) and are not suitable for privacy. RFID tags are good technologies from a privacy perspective because they have no unique attributes of particular users. The decoupling of user's identity and their recognition tag effectively enables us to guarantee anonymity. A user can also have several pseudonyms via multiple tags. Biometric data is more sensitive because if it were extracted from the system, it would provide outsiders with cues to this individual's identity. RFID tags also allow users who are not interested in using the system a way to opt out by not carrying a tag.

C. Encryption Scheme

The basic output of the system is a sequence of files and key-tuples. File F_i is the data logged at time i. Associated with F_i is a tuple of user keys $(k_{i1}, ..., k_{in})$, denoted as *key set*, which determine who has access to this file. The length of the tuple of user keys, n, is always fixed for that environment. The value of n is larger than the maximum number of users who could plausibly be in the environment at the same time. Using a fixed number of keys is necessary if we would like to hide the number of users who are actually in the room at a given time. This many keys may seem extravagant, but in our smart room application, the key set is much smaller than its data file.

Encryption on user data is performed in the following steps:

1) The system performs appropriate key generation algorithm such as ANSI X9.17 [11] and obtains a sequence of pseudorandom keys $d_1, d_2, ...$ One key, denoted d_i, will be selected from the sequence to be used to encrypt file Private key encryption is used for efficiency. Each key will be used only once and the system will perform key generation periodically when the pool of keys is exhausted.
2) d_i is then encrypted with the public keys of the m people in the room (determined by recognition tag reader). These encrypted keys are placed in m locations among the n key positions in the key set, in pseudo-random fashion. The other $n - m$ key positions are filled with random numbers.
3) Users who were in the room can recover the keys and review the video while they were in the room.

Although there are n keys associated with each file, it is not necessary to search through them all in steps 2 and 3. We use a family of n hash functions $h_1, ..., h_n$ to perform a pseudo-random search. At step 2, the system places user j's key in the position specified by one of the hash functions applied to the encrypted image and the user's public key. The first hash function which hashes to a free key location is used. If we assume that n is at least twice the maximum number of people in the room at one time, then at least half these locations will be free, and on average only $\log_2 m$ such steps will be needed to find a free location. The same method is used to retrieve the key at step 3. This scheme is illustrated in Figure 1.

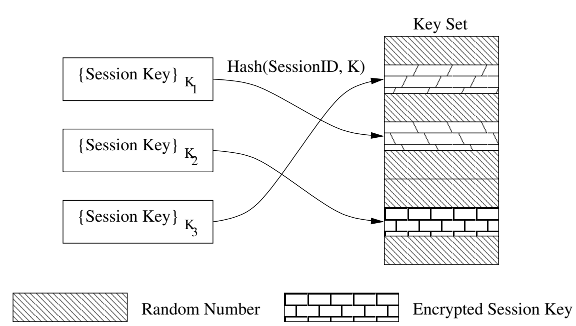

Fig. 1. Key Set Construction

D. Master Key Sharing

It is sometimes necessary for a few privileged parties, e.g., police, to access data stored in the system. However, it is not desirable that a single party be granted full access right to the data since there is a danger of a malicious power party misusing his privilege and compromising users' privacy. Our solution to this dilemma is a shared master key and threshold decryption scheme. The master key consists of a globally known El-Gamal public key and a matching private key that is not held by any single party but instead secrete shared among a number of authorized "masters". Each file's encryption/decryption key d_i is encrypted with the public master key. A group of masters whose number must exceed a pre-specified threshold can collaborate to retrieve d_i and access the data[1]. Pedersen's key generation protocol [12] or its variants/enhancements [13][14] can be used to securely generate the public key and distribute the secret shares of the private key among participants

E. System Architecture

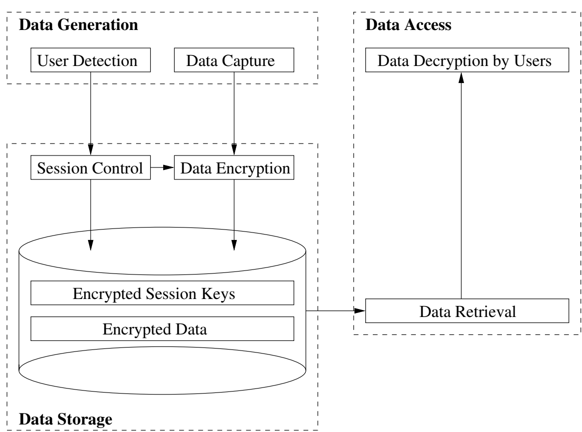

Fig. 2. System Architecture

Putting together all the functionalities described above, our system is organized as shown in Figure 2 and consists of three basic components: *Data Generation*, *Data Storage*, and *Data Access*.Data Generation is the data source that is administrated by the system (as opposite to the users). Besides generating data, it is also responsible for user detection which is essential for establishing the ownership of the data being produced. Data Storage is charged with data encryption and associating users access rights. Data Access is the part that services encrypted data to clients. Since our scheme embeds users' access rights in the encrypted data, this part will do minimal access control and only services users who have registered their public keys with the system.

[1]Note that since the encryption keys are one-time only and only valid for one session, it is safe to reveal it to the collaborating "masters".

V. PERFORMANCE EVALUATION

We have implemented a prototype system on the smart room testbed to carry out our protection scheme. Our smart room has four cameras as well as one tag reader. Images are typically captured at several images per second.

Our prototype system consists of the following four pieces of software:

1) An FTP server that receives image data from the four video cameras and saves them to disk.
2) A data encryption server that constantly checks for new data on disk and encrypts them.
3) A user data server that serves the encrypted data to users upon request.
4) A proof-of-concept client program that continuously requests the most update-to-date data from the user data server, tries to decrypt them on the fly, and displays the recovered images on the screen (if the decryption is successful); thus if the user has legitimate access to the data, he should be able to see a smooth video replay of what's being captured in the room.

Conceptually the first two programs should be merged into one; we did not implement this due to time constraint. Instead, we simulated the effects of one single integrated data receiving/encryption server by deleting the plain text files after we have generated the encrypted version. Nevertheless, we believe that an integrated server is more in line with our security principles and will be implemented in future versions of our system. We used Crypto++ Library™ 5.0 [15], an open source crypto library written in C++, for our encryption and decryption functions.

The encryption server is the crucial part of the system and will be discussed in detail below. Logically it consists of two parallel modules: 1) session control and 2) data encryption. The session control module, a.k.a. the session manager, monitors the users (and their public keys) in the current session and generates session keys. It is notified of user arrivals and departures by the RFID tag reader. A session is defined as a short duration of activities with a fixed set of users. Users' access rights to the data generated at that time remain invariant during one session but would change across sessions. Thus whenever a user enters or exits the room, or whenever a periodic timer expires, the session manager destroys the old session and its associated state, and creates a new session and a new random session key.

The data encryption module monitors the arrival of new image data from the cameras (by periodically checking for new files in a particular file system directory in our case), and encrypts those data with the current session key. We chose Triple-DES with 192-bit keys as our data encryption algorithm. In addition, the module encrypts the session key, which is first prepended with a watchword, with the public key of each user in the current session. We used the RSA public key encryption algorithm for this purpose. The encrypted keys are hidden in the fixed-size key set as described in IV and stored together with data.

We have run several tests on our system. One involves two users simultaneously requesting latest video captures while they are moving in and out of the room randomly. The results are encouraging. Our system can detect the changes of their presence on time and reflect them with the changes of access rights. To determine the throughput of our system, we feed the system with a stream of files of fixed sizes arriving at high speed and measure the time it takes to process them. The experiments were run on a PIII 900MHz machine running Linux 2.4.18 Kernel. We did not account for the cost of generating and encrypting session keys in our experiments because these operations are performed only once per session and are dominated by the cost of encrypting the data files. Figure 3 shows the system throughput with different image file size. As our processing time includes two disk I/Os (one read and one write) as well as the encryption, the throughput improves as the file size increases. With a 1MB file size, the system can achieve a throughput of 2.07MBps while with 8KB file size the throughput is about 1.66MBps. Assuming a capture rate of 20 files per second, our system can support up to 10 cameras. These are not optimal figures, although more than enough for our current needs, but there is plenty of room for improvement. First, the encryption algorithm we use, Triple-DES, is very strong and expensive. Other secret key encryption algorithms exist that can yield much higher throughput. As documented in the Crypto++ 4.0 Benchmarks (http://www.eskimo.com/ weidai/benchmarks.html), DES yields almost 3 times throughput as Triple-DES, Blowfish 4 times, Twofish 6 times, and ARC4 more than 13 times. In cases where the strong property of Triple-DES is not required, these algorithms can be used for higher throughput. Second, we expect the throughput to go up with an integrated receiving/encryption server, as we can eliminate one disk I/O from the critical path. Third, code optimization has yet to be done. Finally, we can always use higher-grade hardware to improve performance.

VI. Toward Trustworthy Environments

The scheme we described in previous sections suffers from a significant weakness: the data is safe only after the server performs the encryption scheme *faithfully*. Therefore, without additional proof, the trustworthiness of the

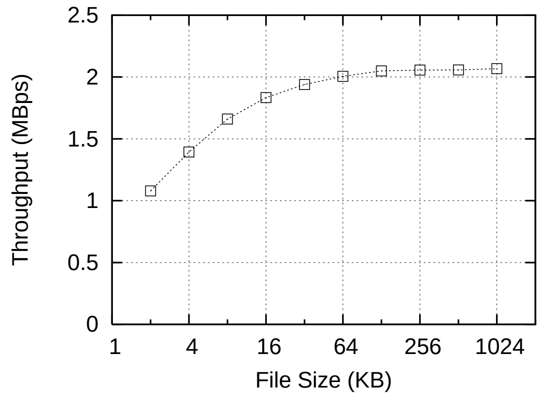

Fig. 3. System Throughput

system depends on the prior trust that users place on it. This means the users have to trust the system in order for it to be trustworthy. This seems to be a hopeless loop. Fortunately there are solutions to this predicament. We believe verification is the way out. Now users do not have to "blindly" trust the system. Rather the system is designed in such a way that it provides "hooks" where users or a third party authority can insert verification procedures and devices to test compliance with any stated privacy policies and practice. We believe that verification is also necessary for ubicomp environments simply because there is no other way to cultivate trust from users who are not affiliated with the owners of the environments.

A. Data Transparency Principle

Verifying a system's compliance with any stated privacy policy remains one of the biggest challenges with ubicomp environments. While there can be no purely technical solutions to this problem, technical means can go some ways to making such verification a realistic possibility. Langheinrich's principle 1 suggests legal sanctions against "covert" monitoring. An unapproved physical device could be detected in a variety of ways, and constitutes a smoking gun for the prosecution. We believe that cryptographic techniques can go a long way to making up the legal slack in addressing principles 2-6. We propose another principle which is

Data Transparency: Encrypted data recorded or transmitted by a ubicomp system should be easily observable. Where possible, the data itself should demonstrate compliance with stated principles.

It may seem paradoxical that easy observability could help enhance privacy. But the information protections provided by well-implemented cryptography are much more reliable than access restrictions using the operating system or network routing. This is especially relevant in ubicomp environments where maintaining correct access or capability information is extremely difficult due to the large amount of data involved and the scale of user base and the requirement for efficient data sharing. The scheme we described in previous sections effectively embeds access rights within the data thus relieving the system of the burden of maintaining complicated access control scheme. Making the encrypted data observable will not harm privacy; on the contrary it makes it possible to implement the verification machinery.

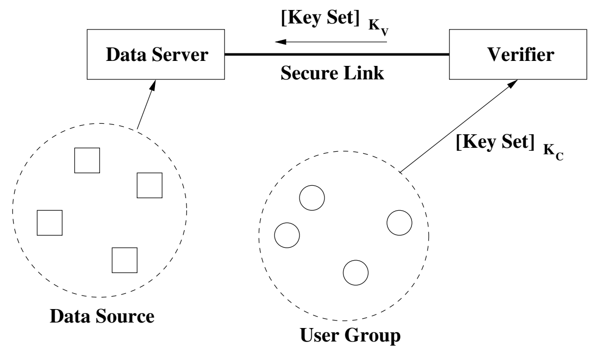

Fig. 4. Verification Architecture

B. Verification Mechanisms

How verification works is the subject of our current research. There are two directions we are investigating. One relies on Zero-Knowledge Proofs (ZKPs). ZKPs allow an agent A to prove to an agent B that A has information that would be very hard for B to compute, without disclosing that information ([16]). More concretely in [1], we used ZKPs to prove that private user data was validly generated without disclosing the data. In our smart room setting, ZKPs could be used to prove that the key set contains only keys for users who were in the room at the time (or rather, that this key set agrees with the set of users encoded in the log files). That is, the system cannot cheat by making data accessible to others without leaving easily-discoverable clues. Once monitoring can be done openly, monitoring equipment could be connected on wired or wireless network links, or run as scanning processes on storage media like disks and flash RAM without fear of privacy compromise.

Another direction takes a system approach and involves the participation of a third party. The third party is generally more trusted because of its neutrality or special position (e.g., government agency). It provides hardware and software that monitor the operation of a ubicomp environment and, periodically, inspects and evaluates the security and privacy compliance of the system. The ubicomp system obtains a certificate once it passes the evaluation and posts it at some noticeable place so the users who enter the environment can be reasonably assured of their data safety and privacy guarantee.

In the second approach, we have devised an architecture, as shown in Figure 4, that addresses the shortcomings of our original protection scheme and incorporates verification and monitoring into normal system operation. This new system includes a third party verifier machine that is the only gateway to the outside world for the data server. The link between the data server and the verifier is assumed to be secure and tamper-evident (This can be achieved by placing the verifier close to the data server and having the verifier provider inspect and seal the link). The verifier has a secrete key burned into its hardware. It acts as an application level filter and only allows data that are not tampered to go out. To illustrate how the system works, let us consider the issue of key set honesty. In the following discussion we assume the clients have some computation power. This is not too restrictive in many situations and our system can still function without it by moving some operations to the verifier.

As described in Section IV, the key set is used to store session key encrypted by the public keys of the users in the room. It is possible for the data server who, in our original scheme, constructs the key set, to insert a copy of the session key encrypted by some other person's public key thus giving him access. This can be avoided in our new architecture by letting clients construct the key set: the clients generate the session key and construct the key set. They will make sure that the non-used slots in the key set are filled with random numbers.

Now we have a few choices: (1) Let the clients carry the key set. It is then secure but inconvenient for the clients. They not only have to carry the key sets for all their sessions (and remember the association between key sets and data files) but also face the risk of losing access since client machines cannot be assumed to be reliable. (2) Clients can upload the key set to the data server. The server can still try inserting another key into the key set but this has a non-trivial probability of being detected ($p = m/n$ where m is number of users and n is number of slots in the key set) because it may overwrite the key of one of the users. When the user fails the decryption he detects this foul play. (3) Clients can upload the key set to the verifier together with some proof of its authenticity (e.g., signature of one of the clients). The verifier can verify this authenticity and protect the key set by MAC using its key and then store it back to the data server. When the data server sends out key set, the verifier will check the authenticity of the key set and only allow authentic key sets to be sent out.

Our new architecture takes the third approach since it involves active verification in normal operation and is most secure. One problem with it, however, is that it does not verify data. The data server can encode the session key in data and leak it out. This is a hard problem because there is no way to verify data. Clients can generate the session key and the key set so clients and the verifier can be sure they are good. However, in many cases even clients cannot verify their data (e.g., the server can alter some bits in an image file and embed the session key in it. Even the clients cannot detect this.).

One way to address this is to let the verifier do the encryption. Data server will forward data to the verifier who will generate a session key and encrypt the data and sign it and store back to data server. Data server will never see the keys. And the verifier will only let data with right signature or MAC go out.

Again there are unsolved issues with this approach. The most obvious question would be: are we simply moving trust away from the ubicomp environment and putting it on the verifier? This is a legitimate concern and we do not claim to have found a complete solution to ubicomp data security. However, we have some arguments for our approach:

1) For users to use any system, trusting at least some part of it is inevitable;
2) The verifier is more trustworthy because it is from a third party who does not understand the data so does not know how to profit from them (therefore has less incentive to steal them)

It is unlikely that any scheme can prevent all types of cheating. We believe ours could certainly raise the technical bar for cheating, and possibly impose strong constraints on the rate of information leakage. This should be enough to reduce the frequency of non-compliance significantly.

C. Guest User Privacy

Our scheme does not address the privacy of "guest user" explicitly. The system cannot detect the presence of anyone without a recognition tag thus cannot enforce protection on his data. There are two possibilities in this case: the guest could be either along or with some registered user(s). In the first case, the guest's data is secure because no one, except the masters, can access it since no one can access the decryption key. In the second case, the registered user(s) can access the data because the system considers them to be the owner(s). We believe this is a valid assumption and also socially acceptable: if one is with some one else at some time, he should expect the other(s) to be aware of his presence and remember it later. Stricter protection on guest data can be enforced by adjustment to the scheme (e.g., stop video taping if one user explicitly requires so) if desired.

VII. Conclusions

As many researchers in Ubiquitous Computing have noticed, in order for Ubiquitous Computing to be really beneficial and socially acceptable, user privacy has to be considered carefully at early stage of system design. In this paper, we argue that the essence of preserving user privacy is protecting user data and propose two design principles. The "data discretion" principle stipulates that access to information stored in a system should only be granted to individuals who would have access to the data in the "real-world". Explicit notion of ownership should be established as the information is generated to determine access right. The "data transparency" principle states that, rather than trying to enhance privacy by hiding the existence of information or communication, a system should rely on well-implemented cryptography for data protection and make the recording and transmitting of encrypted data observable. Only when the usage of data is made open can the system perform effective monitoring to enforce compliance with privacy policies. We consider this to be a very important step toward building trustworthy environment.

References

[1] J. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002, pp. 45–57. [Online]. Available: http://citeseer.nj.nec.com/canny02collaborative.html

[2] "Philips i-code rfid tag reader," http://www.semiconductors.philips.com.

[3] L. Cranor, M. Langheinrich, M. Marchiori, and J. Reagle, "The platform for privacy preferences 1.0 (p3p1.0) specification," W3C Recommendation, April 2002. [Online]. Available: http://www.w3.org/TR/P3P/

[4] G. D. Abowd and E. D. Mynatt, "Charting past, present, and future research in ubiquitous computing," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, pp. 29–58, 2000. [Online]. Available: http://citeseer.nj.nec.com/abowd00charting.html

[5] J. Al-Muhtadi, R. Campbell, A. Kapadia, D. Mickunas, and S. Yi, "Routing through the mist: Privacy preserving communication in ubiquitous computing environ-ments," in *International Conference of Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July 2002.

[6] "Anonymizer," http://www.anonymizer.com.

[7] V. Bellotti and A. Sellen, "Design for Privacy in Ubiquitous Computing Environments," in *Proceedings of the Third European Conference on Computer Supported Cooperative Work (ECSCW'93)*. Kluwer, 1993, pp. 77–92. [Online]. Available: http://citeseer.nj.nec.com/article/bellotti93design.html

[8] M. Langheinrich, "Privacy by design – principles of privacy-aware ubiquitous systems," in *Proceedings of Ubicomp 2001*, ser. Lecture Notes in Computer Science, G. Abowd, B. Brumitt, and S. Shafer, Eds., vol. 2201. Springer, 2001, pp. 273–291. [Online]. Available: http://citeseer.nj.nec.com/491722.html

[9] J. Canny, "Collaborative filtering with privacy via factor analysis," in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Tampere, Finland: ACM Press, 2002, pp. 238–245.

[10] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001. [Online]. Available: citeseer.nj.nec.com/goldberg00eigentaste.html

[11] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, ser. CRC Press Series on Discrete Mathematics and Its Applications. CRC Press, 1996.

[12] T. Pedersen, "A threshold cryptosystem without a trusted party," in *Proceedings of EUROCRYPT '91*, ser. Springer-Verlag LNCS, vol. 547. Springer, 1991, pp. 522–526.

[13] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Lecture Notes in Computer Science*, vol. 1592, pp. 295–310, 1999. [Online]. Available: http://citeseer.nj.nec.com/article/gennaro99secure.html

[14] P.-A. Fouque and J. Stern, "One round threshold discrete-log key generation without private channels," *Public Key Cryptography*, pp. 300–316, 2001.

[15] W. Dai, "Crypto++ Library™ 5.0." [Online]. Available: http://www.eskimo.com/w̃eidai/cryptlib.html

[16] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, vol. 7, no. 1, pp. 1–32, 1994. [Online]. Available: http://citeseer.nj.nec.com/goldreich94definitions.html