Context-Related Access Control for Mobile Caching

Zhi Xu[1], Kun Bai[2], Sencun Zhu[1], Leslie Liu[2], and Randy Moulic[2]

[1] Pennsylvania State University, University Park PA 16802, USA
{zux103,szhu}@cse.psu.edu
[2] IBM T.J. Watson Research, 19 Skyline Drive, Hawthorne NY 10532, USA
{kunbai,lesliu,rmoulic}@us.ibm.com

Abstract. Mobile caching is a popular technique that has been widely applied in mobile applications to reduce the bandwidth usage, battery consumption, and perceived lag. To protect the confidentiality of cached data, the data with sensitive information has to be encrypted as it is cached on mobile devices. Currently, several mobile platforms provide encryption utilities which allow mobile applications to encrypt their local caches. However, existing encryption utilities are too coarse-grained and not directly applicable to dynamically enforcing fine-grained context-related access control policies in context-aware mobile applications.

In this paper, we first show the necessity of new encryption schemes in context-aware mobile applications by examples, and then propose three encryption schemes for enforcing context-related access control policies on cached data. The proposed encryption schemes adopt different cryptographic techniques. By comparing the cache hit rate and communication gain, we analyze the impact of applying the proposed schemes to the efficiency of the existing mobile cache management algorithm in context-aware mobile applications. Further, we evaluate the performance of these schemes through extensive simulations, and suggest the suitable application scenarios for each scheme.

Keywords: Context-related access control, mobile caching, data encryption schemes, context-aware mobile applications.

1 Introduction

1.1 Mobile Caching

Mobile caching is one of the most widely used techniques in web browsers, streaming media applications, and data access applications on mobile devices [1] [2] [3]. Caching recently used data (e.g. routes, pictures of sights) on a mobile device can help the mobile device to reduce the bandwidth usage, battery consumption, and perceived lag.

As most third party mobile applications are only allowed to implement their caches in the application space, one security concern for these mobile applications is the confidentiality of cached data on mobile devices. The attacker may be

M. Rajarajan et al. (Eds.): SecureComm 2011, LNICST 96, pp. 389–408, 2012.

able to access the content of cached data easily if it is stored in plaintext. For example, it has been shown in [4] that, all iPhones, iTouchs, and iPads running iOS 4.0 or later versions log the user's location information in plaintext to a *consolidated.db* file. If the attacker can have access to an iPhone/iTouch/iPad or its synchronized Mac/PC, he can easily map the movement of device using the cached information in *consolidated.db* with tools, like *iPhone Tracker*.

To protect the confidentiality of cached data, the straightforward way is to encrypt the sensitive data as it is cached on the mobile device. Several modern mobile platforms provide encryption utilities which allow mobile applications to encrypt their local caches, for example the data protection features in iOS 4, the password-based encryption feature in BlackBerry OS, and the *EncryptedLocalStore* class in Adobe AIR.

1.2 Context-Aware Mobile Applications

In context-aware mobile applications, the application interacts with the user according to its current *context*, which includes the current location of the mobile device, the current state of the user, the current time, etc. [5] [6]. With the advances in portable devices and sensors, many context-aware mobile applications have been introduced. For example, [7] proposed *Smart Signs*, a context-aware guidance and messaging system providing guests customized route information; [8] proposed a framework for *mHealth* in which the context-related policies are applied when the physician attempts to access the patient's Electronic Patient Record (EPR); and [9] presents a *mobile tourism* application, TIP, which delivers information about sights based on the user's current context.

Obviously, adopting mobile caching technique can also help improve the quality of service in these context-aware mobile applications. However, to enforce context-related access control policies on cached data, the existing encryption utilities are not directly applicable due to several unique challenges.

First, the data access policies to cached data (i.e., data file) in context-aware mobile applications are fine-grained and context-related. Briefly, each data downloaded in the cache is associated with a unique context-related access control policy. Different data may have different access control policies. The cached data accessible in the current context may no longer be allowed to access when the context changes. Mostly, existing encryption utilities are coarse-grained and all data cached in one application are encrypted with the same key. This key is usually generated basing on the application ID and user ID. Therefore, additional extensions are required to suit for context-related policy enforcement.

Second, access control policy enforcement in context-aware mobile applications must be capable of reacting dynamically to the changes of context at runtime, while keeping the efficiency of caching. Existing encryption utilities only support one single context in the cache. When the context changes, the cached data must be cleared. Especially, in mobile applications, the context of a mobile device may change frequently. Erasing cached data whenever context changes may greatly affect the efficiency of mobile caching. Otherwise, if the mobile

application is compromised, all cached data downloaded in both current context and previous contexts will be leaked.

In this paper, we make the first effort to analyze the impact of enforcing context-related access control policies on cached data to the efficiency of mobile caching in context-aware mobile applications. Specifically,we present three encryption schemes using different cryptographic techniques: *Flush Scheme*, *Context Based Encryption (CBE) Scheme*, and *Attribute-based Encryption (ABE) Scheme*. These schemes differ in the strategy to manage the cached data according to context-related access control policies. Among these schemes, our experiment results demonstrate that the *CBE* scheme is most suitable for mobile applications in which the user is usually associated with a static set of contexts and there is little data sharing among different contexts. For example, in a mobile lab application, a scientist is assigned different data access privileges when working on different projects and the project assignment does not change frequently. The *ABE* scheme works best in mobile applications where the user's context changes frequently and some cached data are accessible in different contexts. For example, in the *mHealth* application, the context of a physician may change frequently and unpredictable depending on the patient he is treating.

2 An Example of Context-Aware Mobile Health Information Application

In this section, we present a context-aware mobile health information application, shown in Figure 1, to explain the necessity of applying context-related access control policies on cached data. Also, we show that new encryption schemes are needed to provide confidentiality protection to cached data while allowing the mobile application enjoying the benefits of mobile caching.

As shown in Figure 1, Dr. House is a physician who has a mobile device (e.g. iPad) with a context-aware mobile health information (MHI) application installed on the device. This MHI application allows Dr. House to download and read documents on the mobile device via the application's user interface. These documents are stored in the hospital's content server and protected by context-related access control policies. In this application, the context is determined the status of user, for example, the task Dr. House is performing, the current indoor location of mobile device, the patient who is being treated.

The red trace in Figure 1 shows the trace of Dr. House during a typical workday. In different contexts during the trace, Dr. House will be assigned different privileges by the authority. Here we present two types of documents with different context-related access control policies to show the necessity of applying fine-grained access control on the mobile cache in this application.

One type of documents is the patient's *Electronic Patient Record (EPR)*. To protect an individual patient's privacy, a physician is allowed to read the patient's EPR if and only if he is treating this particular patient in the patient's room. For example, in the first visit to Ellen, the MHI application downloads Ellen's EPR onto the mobile device and saves it in the cache. When Dr. House leaves

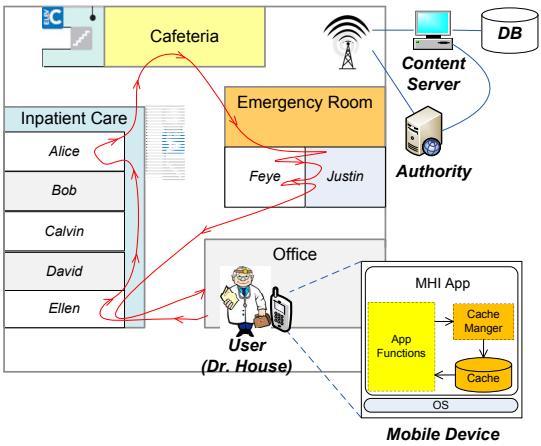


Fig. 1. A mobile health information system

Ellen's room, the Ellen's EPR may stay in the cache but can not be accessed any more because of context change. When Dr. House visits Ellen again, the MHI application can display Ellen's EPR by reading its copy in the cache.

The other type of documents is the *On-Duty Notes* which contains instructions of standardized operations in the hospital. Different to EPRs, on-duty notes are less sensitive and it can be accessed in different contexts as long as Dr. House is within the hospital. Thus, once downloaded into the cache, Dr. House is allowed to access the copy in the cache during the whole trace.

By the comparison of EPRs and on-duty notes,we show that different data cached by context-aware mobile applications on mobile devices may have different access control policies. Thus, fine-grained access control mechanisms are required to enforce their context-related access control policies on cached data.

3 Models and Assumptions

3.1 Mobile Caching Model

The *Mobile Device* (e.g. a smartphone) is connected to a *Content Server* in the client-server manner through wireless connection. In the mobile device, an *Application Cache* is implemented as a part of application. It contains two components: the *Cache* is the local storage keeping the cached data; and the *Cache Manager* is the component managing the cached data. Functions of *Cache Manager* include cache replacement [10], cache invalidation [11], etc. At the server side, we assume that an access control system has been deployed on *Content Server* to guard the data access request from mobile devices to DB. The *Authority* is in charge of user authentication as well as maintaining context-related data access policies. This generic network model has been applied in many mobile information systems, such as [12].

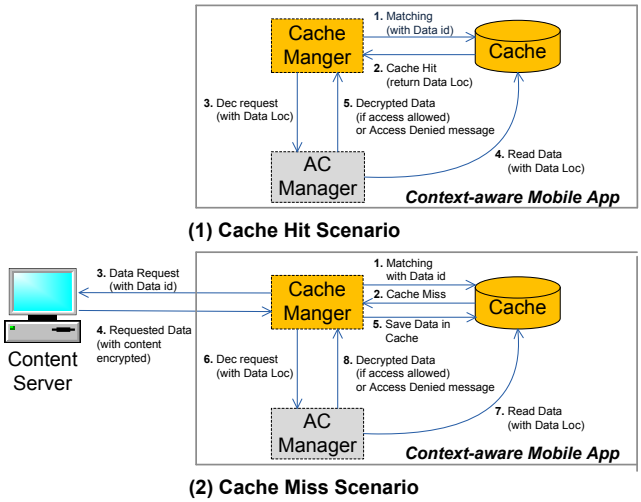


Fig. 2. Basic workflows of proposed system design

In Figure 2, we illustrate the workflow of mobile caching with enforcement of context-related access control policies. We explain the workflow as follows, *Cache Hit Scenario:* If the requested data is contained in the *Cache* (i.e. Cache Hit), the *Cache Manager* will send the reference of data (i.e. the location of encrypted data in the *Cache*) to the *AC Manager* for decryption. The *AC Manager* will be able to decrypt this data if and only if the data access is allowed by the context-related access control policies associated with this requested data. *Cache Miss Scenario:* If the requested data is not contained in the *Cache* (i.e. Cache Miss), the *Cache Manager* will send a data request to the *Content Server*. If access granted, the *Content Server* will encrypt the requested data and send the ciphertext back to the *Cache Manager*. The *Cache Manager* will first save the received data in *Cache* and then request *AC Manager* for decryption.

3.2 Trust Model

In this paper, we trust the integrity of mobile platform and the context-aware mobile application. Specifically, we assume that the mobile application and the mobile platform will perform correctly as required. Also, we assume that the authority is aware of the current context and the context change of mobile device. In Dr. House's example, this assumption means that the attacker (including Dr. House) can not fool the *Authority* with fake context information (e.g. the current location of device) so as to get desired data access privileges. Various location identification (e.g. GSM/3G technology [13]), location tracking [14], location verification techniques (e.g. Echo protocol [15]) can be applied.

We also assume that the compromise of mobile application or mobile platform will be detected within the context in which the compromise happened. Due to the character of mobile devices, such as easy to steal, the attacker may physically possess the mobile device and compromise not only the mobile applications but also the mobile platform. In this case, it is inevitable that the attacker will have access to all information stored on the mobile device, including the decryption keys. However, various techniques can be applied to the integrity measurement insurance and compromise detection, such as such as Trusted Mobile Platform [16], integrity measurements [17] and hypervisor based isolation [18].

3.3 Adversary Model

In this work, the adversaries are nonconforming or curious users who try to bypass the context-related access control policy enforcements and access the content of data stored in the mobile cache. As the application cache is usually implemented in application space, we assume that the adversaries can read and copy the encrypted data in the cache. For example, the adversaries may plug the smartphone to a desktop and copy all the content in the cache storage to the desktop for analysis. However, without the correct decryption key, the attacker cannot get the corresponding ciphertext.

When a data item is required and there exists a cache, the mobile application will always check if there is a copy of requested data before sending a request to the remote content server. Therefore, we assume that the attacker may attempt to access data cached in the previous context by some featured functionalities, such as the "go backward" button and "view history records" function.

3.4 Design Rationale

To enforce the context-related access control policies, one thought is to implement a reference monitor for mobile cache within the mobile application. However, implementing such a reference monitor is too complex and impractical. First of all, this reference monitor will bing a high overhead. It has to keep a detailed record of current context of mobile device, and download the associated access control policies for every data within the cache. Second, the implementation is difficult because the reference monitor at mobile application has to be the same as the reference monitor that is already deployed at content server. Third, the context-related access control policies themselves are sensitive and some companies do not allow downloading these policies from the authority.

Thus, we propose an approach that applies cryptographic techniques to enforcing context-related access control policies. Intuitively, once a data query is approved, the content server will encrypt the requested data with its associated context-related access control policy and send the ciphertext to the mobile application. The encrypted data can be cached locally on the mobile device. At the mobile application(client) side, the mobile application is given a decryption key generated basing on the current context by the authority. The mobile

application will decrypt data on-the-fly. The plaintext of data only appear in the memory and will be deleted after usage.

Compared to the approach with reference monitor, the mobile application in our approach only needs to maintain a decryption key corresponding to its current context and perform decryption operations. When context changes, the mobile application simply replace the outdated decryption key by the new decryption key of the new context. Moreover, cached data will remain in the cache in the *CBE Scheme* and *ABE Scheme* with even less negative impact to efficiency of existing mobile caching schemes. Details of designs and implementations will be introduced in later sections.

4 Proposed Schemes

To enforce context-related access control policies within the *AC Manager*, we propose three cryptographic schemes: *Flush Scheme*, *Context Based Encryption (CBE) Scheme*, and *Attribute Based Encryption (ABE) Scheme*. These schemes differ in the strategy of policy enforcement and they perform best in different application scenarios. Briefly,

- In the first (simplest) scheme, *Flush Scheme*, the user is only allowed to cache data requested within the same context. When the context changes, all existing cached data will be erased.
- In the second scheme, *Context Based Encryption Scheme*, the cached data are encrypted basing on the context when they are downloaded from the content server. When the context of mobile device changes, existing data in the mobile cache will not be erased. If the mobile device change from context A to context B and then back to context A, the cached data downloaded previously in context A can still be accessed if it is still in the cache.
- In the third scheme, *Attribute-Based Encryption Scheme*, we adopt the *ciphertext-policy attribute-based encryption* (CP-ABE) technique to further improve the caching efficiency and flexibility by allowing possible sharing among different contexts. For example, suppose that the data m is downloaded in context A, and it is allowed to be accessed by a mobile device in both context A and B according to the context-related access control policy. If the mobile device changes from context A to context B and m is in the mobile cache, the user will be able to access the cached data m.

In the rest of section, we explain the motivation of proposed schemes and their strategies to deal with context changes. Due to the space limit, the detailed encryption and decryption procedures are not presented.

4.1 Scheme One: Flush Scheme

In *Flush Scheme*, the user is only allowed to cache data requested within the current context. The *Flush Scheme* utilizes the context-based access control enforcement at the *Content Server*. If the downloading request of a data is allowed

at the *Content server*, the access to its cached copy of data should be allowed too in the same context. However, whenever the context changes, the *AC Manager* will erase all existing cached data.

The Flush Scheme can be implemented using *Secret Key Cryptography (SKC)*. For each context of a user, the *Authority* generates a random secret key K_{rand}, which will be sent to *AC Manager*. When context changes, the *AC Manager* will replace the old context's decryption key by the one of new context. In any circumstances, the *AC Manager* only keep the decryption key of current context. The cached data may be in the form of $M_{data} = (C_{data}, Time_{data\_exp}, ID_{context}, Hash())$.

4.2 Scheme Two: Context Based Encryption Scheme

In the *Context Based Encryption (CBE) Scheme*, the cached data are encrypted based on the current context of user. When the context changes, the existing data in the *Cache* will not be affected. The deletion of cached data in CBE scheme is managed by the *Cache Manger* according to ordinary cache management schemes, such as Latest Recent Used (LRU) scheme.

The decryption procedure in *CBE Scheme* involves two contexts. One is the context in which the data is encrypted at *Content Server* and delivered to the mobile cache. The other is the context in which the *AC Manager* tries to decrypt the data. In the CBE Scheme, the decryption can be performed correctly if and only if these two contexts are the same.

Here a context can be represented by a set of privileges assigned to this particular user within this context or simply an ID assigned to this context. A user may re-enter the same context multiple times. For instance, in the example of MHI application, whenever Dr. House enters Ellen's room, the context will be the same. Thus, Dr. House will be able to read Ellen's EPR directly from the cache when revisiting Ellen's room, if its encrypted copy is still in the cache.

The *CBE Scheme* can also be implemented using *SKC*. For each context of a user, the *Authority* generates a secret key $K_{context}$ based on the current context of mobile device/user. When the user enters the same context, the assigned decryption key will be the same as well. The cached data in *CBE Scheme* is in the form of $M_{data} = (C_{data}, Time_{data\_exp}, ID_{enc\_context}, Hash())$.

4.3 Scheme Three: Attribute-Based Encryption Scheme

In the third scheme, *Attribute-Based Encryption Scheme*, we adopt the *ciphertext-policy attribute-based encryption* (CP-ABE) technique [19] to further improve the caching efficiency and flexibility by allowing possible sharing among different contexts. In the *ABE Scheme*, a cached data may be accessed in different contexts as long as these contexts satisfy the data's associated context-related access control policies. For instance, in the example of MHI application, if the *On-Duty Notes* is already in the cache, Dr. House will be able to access the cached copy in different contexts during the trace. Because Dr. House is always in the hospital in this example.

Specifically, in the ABE Scheme, the privileges of a user in a context is represented by a set of attributes. In each context, the user will be assigned a decryption key generated by the attributes assigned to the current context. On the other hand, the context-related access control policy associated with one data is represented by an access structure $\mathbb{A}$ on a set of attributes. During the encryption, the encryptor (i.e. *Content Server*) encrypts the plaintext of data with $\mathbb{A}$. In the *AC Manager*, the decryption can be performed correctly if and only if the attributes of user's decryption key satisfy the access structure $\mathbb{A}$ associated with the ciphertext. Various CP-ABE schemes have been proposed, currently we adopt the CP-ABE scheme introduced in [19] to describe how we apply this scheme in our *ABE Scheme*. Specifically, the *Authority* first generates a master key MK and a public key PK. The public key PK will be given to the *Content Server* and *AC Manager* on a mobile device. The master key MK will never leave the *Authority*. When a context starts, the *AC Manager* will receive a decryption key D which is generated based on the master key MK and the set of attributes assigned to the user in this context.

When encrypting data, the *Content Server* encrypts the data content with the public key PA and an access structure $\mathbb{A}$. The ciphertext is now in the form of $M_{data} = (C_{data}, Time_{data\_exp}, ID_{context}, \mathbb{A}, Hash())$. When decrypting data, the *AC Manager* uses its current decryption key D, public key PK, and follows the access structure $\mathbb{A}$. Due to the space limit, please refer to [19] for details of the CP-ABE scheme.

5 Simulation

In this section, we study the impact on existing mobile cache management schemes when applying proposed encryption schemes to enforce context-related access control policies on cached data. Specifically, we measure the changes of efficiency of the underlying cache replacement algorithm that are caused by applying proposed schemes. Efficiency is critical to our proposed schemes. Because enforcing context-related access control policies over the cached data may neutralize the benefits gained by caching. If allowing mobile caching with access control is too costly, people would prefer disallowing caching any sensitive data on the mobile device. Each of proposed schemes has its own pros and cons in terms of efficiency. Details of cost and benefit analysis are presented in appendix.

5.1 Simulation Setup

A Query Model with Context Changes. Existing query models proposed to evaluate mobile caching algorithms do not take context into consideration. Therefore, we present a new query model to simulate the user behavior in the context-aware mobile application.

In our new query model, we use a sequence of ordered queries to represent the data queries issued by one user within a period of time. Each query consists of a data ID, a context ID, and a timestamp, representing respectively the

data requested, the current context, and the current time when the query is raised. The *ThinkTime* between neighbor queries in the sequence by following an Exponential distribution. The timestamp for the first query in the sequence is zero.

To model the database protected by context-related access control policies at content server, we crate a database and divide its data items into different groups. Each group corresponds to one context. To model the case when data may be accessible in two contexts, we randomly select a portion (p) of data in each group and pair them. Those paired data will be considered as the equivalent data items. For each group/context, we generate a set of queries satisfying a zpif distribution on the data in the database. These sets of queries represent the data queries generated by the user in different contexts.

The context change mode is application-specific and is usually derived from traces of real user behavior [20] [21]. In this paper, we consider the general case of context changes. Specifically, we create a *Markov Chain model*, in which the states represent possible contexts of users and the state transitions represent the context changes. Each state has one transition to itself, representing that the next query will stay in the current context, and one transition to any other state in the model, representing the context change. We assume that, if a context change happen, the next query may be in any one state in the model with the same probability. Formally, let S $(s, s' \in S)$ denotes the set of states in the model, the context of current query is at X, the context of next query is at X', and λ be the probability that the s' will state in the same context. Then,

$$Pr(X' = s'|X = s) = \begin{cases} \lambda & if s' = s \\ \frac{1-\lambda}{|S|-1} & s' \neq s \end{cases} \quad (1)$$

Parameters Selection. Four parameters are considered in simulations: *cache size*, *data sharing rate*, *context change rate*, and *time-to-live (TTL)*. *Cache size* represents the resource constraint on mobile devices. *Context change rate* represents the dynamics of user's status and the query pattern of user. *Data sharing rate* describes the characteristic of data in DB. *TTL* defines the maximum length of time a data item is allowed to cache locally. Table 1 presents the settings of other parameters in simulations.

Efficiency Measurements. The efficiency metrics in the study are the Cache Hit Rate (CHR) and the Communication Gain (CG). The cache hit rate is computed by dividing the sum of the queries that are answered using *Cache* by the sum of the total queries in the simulation. The communication gain is computed by the data transmission saved by caching minus the data transmission brought by synchronization between a mobile device and *Authority*. We compare the three proposed schemes under different cache size, expiration time (i.e. Time-To-Live), and context change frequency.

The CG is measured by counting the amount of data downloaded by mobile device with a sequence of queries. In the base case

Table 1. Simulation parameter settings

Parameter	Setting
Query sequence length	1200
Zipf distribution	θ=0.80
Number of context sets	4
Database size each context	1000
Cache size	20, ..., 400
ThinkTime (T_t)	Exponential Distribution (mean=100s)
Data item size (the same size)	1KB, 15KB, 100KB
Data sharing distribution	p=5%

(i.e. disallowing caching), the total amount of data downloaded is denoted by $BaseAmount = (sequence\_length \times data\_size)$. So, we compute CG by $Syn\_Cost + Comm\_Cost - BaseAmount$. The *Synchronization Cost (Syn_Cost)* stands for context change information downloaded from Authority whenever context changes. That is $Syn\_Cost = (number\_of\_context\_changes) \times (data\_downloaded\_per\_context\_change)$; The *Communication Cost(Comm_Cost)* stands for the amount of data downloaded from Content Server with the sequence of queries. That is $Comm\_Cost = |M_{data}| \times (sequence\_length) \times (1 - CHR)$.

For comparison purpose, we ignore the cost for building a secured communication channel between a mobile device and authority. Also, we ignore the context change request message sent from a mobile device to Authority. We just measure the data required to transfer from Authority to a mobile device. Suppose the size of original data is $|P_{data}|$. For encryption and decryption, the Flush and CBE scheme use the AES-128, and the ABE scheme uses the *cpabe* toolkit. In addition, SHA-1 is used for the one-way hash function. For each proposed scheme, we calculate $|update|$ (i.e. the new context information downloaded per context change) and $|M_{data}|$ (i.e. the response from *Content Server* with the ciphertext of requested data). Note that, in ABE scheme, the sizes of decryption key and ciphertext depend on the number of attributes associated with the decryption key and the access structure $\mathbb{A}$, respectively. In our experiments, a decryption is about 23.8 KB with 3 attributes and 94.4KB with 8 attributes. When $\mathbb{A}$ contains 7 attributes, the size of ciphertext will be about 14.4KB larger than the plaintext. When $\mathbb{A}$ contains 15 attributes, the size of ciphertext is about 15.2KB more than the plaintext. For our discussion, we assume that the size of decryption key is 23.8KB (3 attributes case) and the size of ciphertext will increase with 14.4KB (7 attributes case).

5.2 Experiment 1: CHR vs. Cache Size and Data Sharing Rate

In this experiment, we measure the performance of our proposed schemes with different cache sizes. Considering the data sharing, we also measure its CHR with three sharing rates (Hot, Cold, and Random). By sharing, we mean a data

item is allowed to be access by multiple set of contexts. As data queries follow a zipf distribution [22], popular data items are data that are queried most.

- Hot-Sharing: randomly choose 50 (i.e. $5\% \times 1000$) from the top 20% popular data items;
- Cold-Sharing: randomly choose 50 from the 20% of least popular data items;
- Random-Sharing: randomly choose 50 from the whole DB;

As we can see from Figure 3(a), increasing the cache size may help improve the CHR in the proposed schemes. Among these schemes, the ABE scheme always achieves the highest CHR in all cache sizes, and the CBE scheme is close to ABE scheme. Both CBE and ABE scheme have much higher CHR than Flush scheme. Further, when the sharing rate increases, the ABE scheme gains more advantages. For Flush scheme, we observe that the effect of increasing cache size is affected by context changes. As shown in the Figure 3(a), when the cache size increases to a threshold point, the CHR cannot be improved any more. This is because the Flush scheme will empty the Cache whenever there is a context change. When the cache size is big enough for any single context, increasing its size will not improve the performance of CHR.

5.3 Experiment 2: CHR vs. Context Change Rate

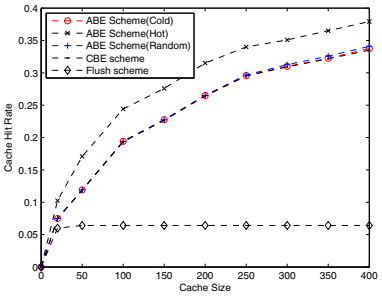
In this experiment, we study the impact of increasing context change frequencies on the CHR with different sized cache. Specifically, we measure CHR with three cache sizes: 400(BIG), 200(MEDIUM), and 20(SMALL). The purpose is to investigate the resistance to frequent context changes in proposed schemes. The scheme resistant to frequent context changes will be suitable for applications in dynamic environments. As the context change rate reflects the user's behavior pattern, the experiment results help the application developer to choose right combination of schemes and cache sizes for different behavior patterns.

From the experiment results shown in Figure 3(b), the CBE and ABE schemes are more resistant to context changes. To both CBE and ABE schemes, the cache size is the dominating factor for CHR. To the contrary, Flush scheme highly relies on the context change rate.

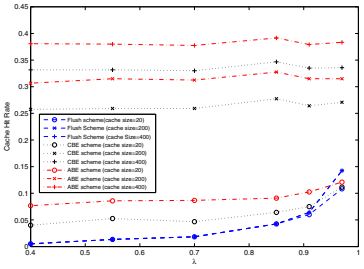
Furthermore, a careful comparison between CBE and ABE schemes shows that the gap between these two are shrinking as the cache size increases. This is because ABE scheme utilizes the space in cache more efficiently than CBE scheme. Without sharing, data shared by multiple contexts may have several copies in the Cache belonging to different context sets in CBE scheme. Therefore, when the cache size is small, ABE scheme achieves more advantages. When the cache size is big, the effect of cache space utilization is reduced.

5.4 Experiment 3: CHR vs. TTL

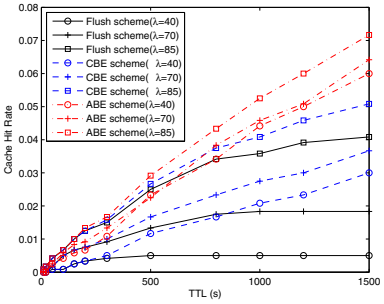
Time-To-Live (TTL) is the duration a data item is allowed to be cached on a mobile device. TTL is determined by Authority when giving the permission assignment, and enforced at Content Server when sending the data. For Authority,



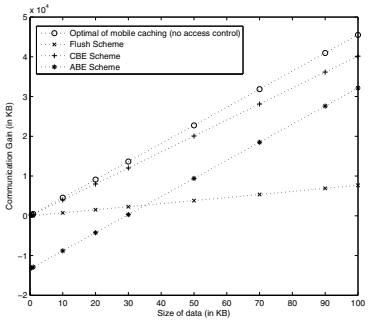
(a) The impact of changing cache size to CHR



(b) The impact of context change rate to CHR



(c) The impact of TTl to CHR



(d) A comparison of CG with various data size

Fig. 3. Evaluation of mcRBAC Schemes

he wants to assign the TTL "just long enough" for future data access. The idea is quite similar to Belady's optimal cache replacement algorithm, which always discards the data that will not be needed for the longest time in the future. But, the *Authority* sets TTL from perspective of security.

In this experiment, given a cache size, we would like to investigate how to choose TTL according to user's query pattern, i.e. the context change frequency. The result will help us (1) understand the impact of TTL with different DB access patterns in proposed schemes; (2) understand how to choose right TTL for different DB types. A comparison combining all three schemes is in Figure 3(c). From the experiment results of ABE and CBE schemes, we see that the CHR keeps increasing as the TTL increases, because both schemes allow accessing data cached in previous contexts. Therefore, keeping data in the cache for longer time will result in higher CHR.

In the Flush scheme, the CHR increases to a threshold and then stops increasing with the increase of TTL. This is because flush scheme will empty the cache when context changes. Thus, the context change rate λ determines the value of threshold in *Flush* scheme. As shown in the figure, the larger λ results

in higher maximum CHR. Therefore, when applying CBE or ABE schemes, one may consider adjusting TTL as a way to improve the efficient of existing cache management schemes, similar to the idea of using Adaptive TTL approach [23] to maintain the cache coherency. when applying Flush scheme, user's query pattern (including context change pattern) should also be considered. A overly large TTL will be a waste in Flush scheme. How to set an optimal TTL for a cached data can be one of our future works.

5.5 Experiment 4: CG vs. Data File Size

In previous experiments, the *ABE* always achieves the highest CHR. However, the size of cipher-text in *ABE* scheme is larger than in the other two schemes. The purpose of this experiment is to show the tradeoff between CG and CHR in different cache sizes. We present the results in Figure 3(d).

In this comparison, the optimal results are generated using the Hypothetical Optimal Scheme with no access control. As shown in the figure, when data size increases, the CG in all schemes increases. When the CG is below 0, it means that it downloads more data than the case without caching. In such a case, it would be better not to allow caching at all.

Flush scheme has very little overhead, however, the gain from CHR is also little. When the size of data increases, the benefits of CHR become greater. CBE scheme has little overhead but moderate CHR. Therefore, its CG is very close to the optimal case. However, as the file sizes increases, the gap between these two will increase because of the differences in CHR; ABE scheme has a high overhead thus not suitable for cases when data size is small. However, when the data size increases, the benefits of CHR starts to beat the overhead. When the file size is greater than 30KB, the CG of ABE climbs to above 0. Also notice that, the gap between ABE and CBE scheme is shrinking, meaning that the advantage in CHR may be more important when the data size is big and the sharing rate is high. If the sharing rate is not high, ABE scheme may not beat CBE scheme by gaining from CHR improvement on allowing data sharing. Therefore, when the data size is small, the Flush scheme and CBE scheme may be more suitable in terms of CG. When the data size is moderate, CBE may be a better choice. When the data size is huge, one may consider the ABE scheme. Because, in this case, achieving better CHR has more direct effect to CG.

5.6 Experiment 5: CG vs. Context Change Rate

In Experiment 2, we have shown that, the ABE scheme has the best CHR when context changes frequently. In perspective of CG, things may be different because the great overhead of context changes in ABE scheme. In this experiment, we study the impact of context change rate in the same setting as Experiment 2, but now we compare schemes from the perspective of CG. The data size in this experiment is fixed as 50 KB.

The result of comparison is shown in Figure 4. From the experiment results, we can clearly see the tradeoff of applying different schemes. The Flush scheme

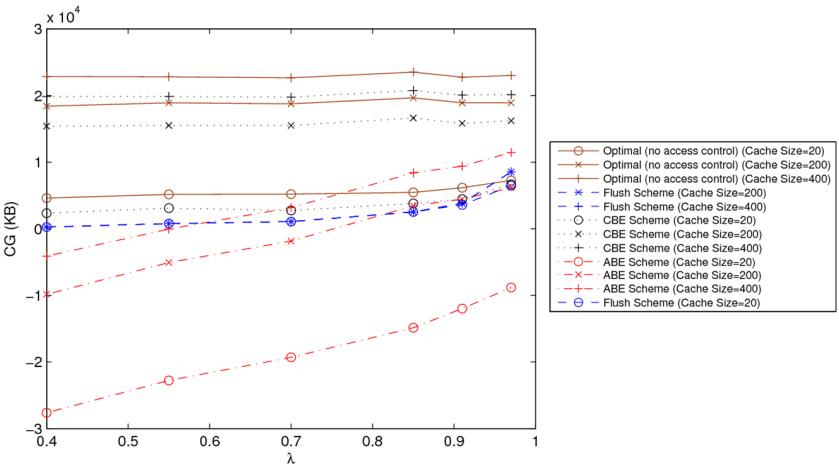


Fig. 4. The impact of context change rate to CG

has the lowest overhead, and yet its CHR is limited. To another extreme, the ABE scheme can achieve highest CHR, and yet its overhead of context change is too high. When context changes frequently or the gain from CHR is too little, the CG of ABE scheme will be much below 0. The CBE scheme provides the best CG in this experiment. As we can see from Figure 4, it is always close to the optimal case. Therefore, when the cache size is small and data file is not big, the CBE scheme would be the best choice. ABE scheme is more suitable for cases with bigger data size. Flush scheme would perform best if the cache size is big and the data sharing rate is small.

6 Related Work

6.1 Context-Related Access Control for Mobile Computing

In [24], a system called *CRePE* (Context-Related Policy Enforcing) is proposed, which extends the permission checking of Android to support enforcing context-related security policies at run-time. Differently, no permission checker is required in the proposed schemes. Instead, we rely on cryptographic techniques and enforcing context-related security policies by carefully assign the user different decryption keys according to their contexts.

Also, in many research works, such as [25, 26], the authors consider context as environment roles and propose context-related access control approaches by extending the RBAC model with spatial and temporal information. For example, [25] proposes a Spatial Role-based Access Control (SRBAC) model which allows the authority to use location information in security policy definitions. [26] proposes a GEO-RBAC model which allows securing the access to spatial data in location-aware applications.

Different to these extended RBAC models, first of all, our work is more data-oriented and focuses on enforcing the access control policy at the mobile device side. Second, we rely on the deployed access control system to detect and enforce the context change (i.e. updating the decryption key). The *AC Manager* does not run as a reference monitor. Third, our focus is not about how the context of mobile device changes. Instead, our proposed schemes focus on mechanisms of enforcing context-related access control policies when the context changes.

6.2 Distributed Data Management

Many cryptographic techniques have been applied to enforce access control policies on distributed data. For example, [27] proposes a Fine-grained Distributed data Access Control (FDAC) Scheme which applies Key-Policy Attribute-Based Encryption (KP-ABE) to protect distributively stored sensed data in wireless sensor networks. Different from the FDAC scheme in [27], the mobile device in our design does not allow to encrypt or publish data. The mobile device only has the decryption function and is limited by decryption key assigned. In addition, [28] and [29] introduce the Attribute Based Messaging (ABM) system which allows the message sender to specify allowed recipients with attribute-based access control policies. Specifically, [29] discusses employing CP-ABE to provide end-to-end confidentiality for ABM. [30] applies the ABM to secure the first response coordination in mobile environment. Different from the push model in ABM, we adopt the pull model in which the data request is generated by the mobile device and responded by the server. Moreover, [29] focuses on recipients classified by attributes. Differently, our focus is the same recipient with changing contexts.

6.3 Access Control on Mobile Devices

Some research works have been done to design access control systems for mobile devices. For example, [31] proposes a mandatory access control (MAC)-based mechanism on cellphone with the purpose of controlling the program accesses to important system resources. [32] proposes the design of a trusted subsystem which can be used to enforce MAC on mobile devices. [33] proposes a *TaintDroid* system, which tracks the information-flow of privacy sensitive data through third-party applications. [34] proposes a *Porscha* system, which enforces Digital Rights Management policies on smartphones. Both TaintDroid and Porscha require to implement a reference monitor within the kernel of Android platform.

In this work, we focus on enforcing context-related access control policies on the application cache only. All components of access control are within the mobile application space and implemented by the developer of context-aware mobile app. Thus, our proposed schemes can be easily implemented on commodity mobile devices with little modification and overhead.

7 Conclusion

We study the problem of enforcing context-related access control on cached data in mobile devices. Specifically, we propose the design of three encryption schemes adopting different cryptographic techniques. We present a quantitative comparison of proposed schemes through analysis as well as simulations. We show an application on commodity smart phones.

In our future work, we are planning to work on two directions: one direction is to apply the proposed schemes to other context-aware mobile applications. The other direction is to look for best cryptographic implementations suitable for proposed schemes on different smartphone platforms.

References

1. Jiang, Z., Kleinrock, L.: Web prefetching in a mobile environment. IEEE Personal Communications 5, 25–34 (1998)
2. Höpfner, H., Wendland, S., Mansour, E.: Data caching on mobile devices - the experimental mymidp caching framework. In: Proc. of the 4th International Conference on Software and Data Technologies (2009)
3. Apple, "Safari developer library: Storing data on the client", `http://developer.apple.com/library/safari/`
4. Allan, A., Warden, P.: Got an iphone or 3g ipad? apple is recording your moves (2011), `http://radar.oreilly.com/2011/04/apple-location-tracking.html`
5. Schilit, B.N., Adams, N., Want, R.: Context-aware computing applications. In: Proc. of The Workshop on Mobile Computing Systems and Applications, pp. 85–90. IEEE Computer Society (1994)
6. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, pp. 304–307. Springer, Heidelberg (1999)
7. Lijding, M., Meratnia, N., Benz, H.: Smart signs show you the way. IO Vivat 22(4), 35–38 (2007)
8. Kyriacou, E.C., Pattichis, C., Pattichis, M.: An overview of recent health care support systems for eemergency and mhealth applications. In: Proc. of 31st Annual International Conference of the IEEE EMBS (2009)
9. Hinze, A., Buchanan, G.: Context-awareness in mobile tourist information systems: Challenges for user interaction. In: Proc. Workshop on Context in Mobile HCI, in Conjunction with Mobile HCI (2005)
10. Johnson, T., Shasha, D.: 2q: a low overhead high performance buffer management replacement algorithm. In: Proc. of the 20th International Conference on Very Large Databases (1994)
11. Cao, G.: A scalable low-latency cache invalidation strategy for mobile environments. IEEE Trans. on Knowl. and Data Eng. (2003)
12. Ferraiolo, D.F., Barkley, J.F., Kuhn, D.R.: A role-based access control model and reference implementation within a corporate intranet. ACM Trans. Inf. Syst. Secur. 2, 34–64 (1999)
13. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Supporting location-based conditions in access control policies. In: Proc. of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, pp. 212–222 (2006)

14. Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: Proc. of the 6th Annual International Conference on Mobile Computing and Networking, MobiCom 2000, pp. 32–43 (2000)
15. Sastry, N., Shankar, U., Wagner, D.: Secure verification of location claims. In: Proc. of the ACM Workshop on Wireless Security (WiSe 2003), pp. 1–10 (2003)
16. N. DoCoMo, IBM, I. Corporation: Trusted mobile platform: Hardware architecture description (2004)
17. Muthukumaran, D., Sawani, A., Schiffman, J., Jung, B.M., Jaeger, T.: Measuring integrity on mobile phone systems. In: Proc. of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, pp. 155–164 (2008)
18. Cox, L.P., Chen, P.M.: Pocket hypervisors: Opportunities and challenges. In: Proc. of the Eighth IEEE Workshop on Mobile Computing Systems and Applications, HOTMOBILE 2007, pp. 46–50 (2007)
19. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proc. of the 2007 IEEE Symposium on Security and Privacy, SP 2007, pp. 321–334 (2007)
20. Chen, G., Kotz, D.: A survey of context-aware mobile computing research, Hanover, NH, USA, Tech. Rep. (2000)
21. Kim, M., Kotz, D., Kim, S.: Extracting a mobility model from real user traces. In: Proc. of the IEEE International Conference on Computer Communications (IEEE INFOCOM 2006) (2006)
22. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: Evidence and implications. In: Proc. of the Conference on Computer Communications (IEEE Infocom 1999) (1999)
23. Cate, V.: Alex-a global file system. In: Proc. of USENIX File System Workshop 1992, pp. 1–12 (1992)
24. Conti, M., Nguyen, V.T.N., Crispo, B.: CRePE: Context-Related Policy Enforcement for Android. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 331–345. Springer, Heidelberg (2011)
25. Hansen, F., Oleshchuk, V.: Srbac: A spatial role-based access control model for mobile systems. In: Proc. of 7th Nordic Workshop on Secure IT Systems (2003)
26. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: Geo-rbac: A spatially aware rbac, vol. 10. ACM (2007)
27. Yu, S., Ren, K., Lou, W.: Fdac: Toward fine-grained distributed data access control in wireless sensor networks. In: Proc. of the IEEE International Conference on Computer Communications (IEEE INFOCOM 2009), pp. 963–971 (2009)
28. Bobba, R., Fatemieh, O., Khan, F., Gunter, C.A., Khurana, H.: Using attribute-based access control to enable attribute-based messaging. In: Proc. of the 22nd Annual Computer Security Applications Conference, pp. 403–413 (2006)
29. Bobba, R., Fatemieh, O., Khan, F., Khan, A., Gunter, C.A., Khurana, H., Prabhakaran, M.: Attribute-based messaging: Access control and confidentiality. ACM Transactions on Information and Systems Security, TISSEC (2010)
30. Weber, S.G.: Securing first response coordination with dynamic attribute-based encryption. In: Proc. of World Congress on Privacy, Security, Trust and the Management of e-Business 2009 (2009)
31. Xie, L., Zhang, X., Chaugule, A., Jaeger, T., Zhu, S.: Designing system-level defenses against cellphone malware. In: Proc. of the 28th IEEE International Symposium on Reliable Distributed Systems, pp. 83–90 (2009)
32. Zhang, X., Seifert, J.-P., Sandhu, R.: Security enforcement model for distributed usage control. In: Proc. of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Sutc 2008 (2008)

33. Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: Proc. of the USENIX Symposium on Operating Systems Design and Implementation, OSDI (2010)
34. Ongtang, M., Butler, K., McDaniel, P.: Porscha: Policy oriented secure content handling in android. In: Proc. of the 26th Annual Computer Security Applications Conference, ACSAC (2010)
35. Bethencourt, J., Sahai, A., Waters, B.: The cpabe toolkit in advanced crypto software collection, `http://acsc.cs.utexas.edu/cpabe/`

A Appendix

A.1 Security Analysis

Inter-Context Compromise Resistance. In both CBE and ABE schemes, data downloaded in previous contexts may remain in the cache. If the mobile application is compromised, the attacker may possess the decryption key of current context. The content of cached data accessible in current context will be leaked inevitably. However, in both schemes, the data leakage is limited to only cached data that are accessible in the current context. Because the compromise will be detected by the authority within the current context, only the decryption key of current context is stored in the mobile application. Therefore, the attacker can only possess the decryption key of current context.

Collusion Resilience. According to the adversary model, the attacker may possess multiple mobile devices. In this case, it is critical for access control schemes to be resilient to collusion attacks. That is, the attacker should not be able to derive new decryption keys by keys he possessed. To defend against collusion attacks, in the *Flush Scheme*, the context's secret key $K_{context}$ is randomly chosen for each context thus collecting multiple keys will have no use at all. Similarly, in the *CBE Scheme*, the keys are randomly generated basing on a set of contexts instead of for every single context. Thus, the attacker will not be able to use multiple decryption keys of different users or the same user to generate a new decryption key. In the *ABE Scheme*, the collusion resilience is provided by the CP-ABE scheme. For example, [19] adds randomness in the data encryption and decryption key generation to prevent collusion attacks.

A.2 Computation Overhead Analysis

The major computation overhead is caused by performing decryption by *AC Manager*. If there is a cache hit and the data is allowed to be accessed, both *Flush Scheme* and *CBE Scheme* need one round decryption operation with a secret key. If there is a cache hit by data id but the access is denied, both *Flush Scheme* and *CBE Scheme* result in access denied by a simple value-based comparison. More expensively, *CBE Scheme* requires a decryption attempt to reveal the feasibility of decryption. Because the *ABE Scheme* requires to perform a series of decryption operations following the access structure $\mathbb{A}$. According to the

measurements in [19], the decryption workload depends greatly on the particular access tree $\mathbb{A}$ and the set of attributes involved in the decryption. From the perspective of decryption algorithm implementation, the efficiency of elliptic curve based operations is the key for the decryption speed.In [27], the author presented an efficient implementation of elliptic curve based operations on sensors with low computational capacity. Currently, the implementation we are using is the *cpabe* toolkit implemented at *Advanced Crypto Software Collection (ACSC)* developed by John Bethencourt, et al. [35].

A.3 Implementation Complexity Anslysis

Flush Scheme and *CBE Scheme* depend on SKC based decryption which is easy to implement and has many efficient implementations already. The *CP-ABE*, on which *ABE Scheme* relies, is relatively new compared to SKC and only have several implementations provided by research groups.

A.4 Efficiency Metrics

Efficiency is critical to our proposed schemes. Because enforcing context-related access control policies over the cached data may neutralize the benefits gained by caching. If allowing mobile caching with access control is too costly, people would prefer disallowing caching any sensitive data on the mobile device.

Cache Hit Rate (CHR). The Cache Hit Rate (CHR) is represented by the percentage of data accesses that results in mobile cache. It is computed by dividing the sum of the queries that are answered using *Cache* by the sum of the total queries in the simulation. Other performance metrics, such as query delay, throughput, and data communication cost, all have a strong relation with CHR.

Communication Gain. Communication Gain (CG) measures the benefits of applying a proposed scheme in terms of data downloaded. Applying access control may require extra data downloaded, because of the synchronization between the *Authority* and the mobile device. To measure CG of a proposed scheme, we count the overall data downloaded with a sequence of queries and then compare it with that in the case without caching (i.e. *base case*). If the CG of a scheme is negative, it means that applying this scheme will need to download even more data than the base case.